

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	5
1.3.1	Argument Promotion	5
1.3.2	NaN Arguments	5
1.4	Implementation	6
1.5	Testing	6
1.6	General Bibliography	6
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)	39

8.2.2.48	laguerrel(unsigned int __n, long double __x)	39
8.2.2.49	legendre(unsigned int __l, _Tp __x)	39
8.2.2.50	legendref(unsigned int __l, float __x)	40
8.2.2.51	legendrel(unsigned int __l, long double __x)	40
8.2.2.52	riemann_zeta(_Tp __s)	40
8.2.2.53	riemann_zetaf(float __s)	41
8.2.2.54	riemann_zetal(long double __s)	41
8.2.2.55	sph_bessel(unsigned int __n, _Tp __x)	41
8.2.2.56	sph_besself(unsigned int __n, float __x)	42
8.2.2.57	sph_bessell(unsigned int __n, long double __x)	42
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)	43
8.2.2.60	sph_legendrel(unsigned int __l, unsigned int __m, long double __theta)	43
8.2.2.61	sph_neumann(unsigned int __n, _Tp __x)	43
8.2.2.62	sph_neumannf(unsigned int __n, float __x)	44
8.2.2.63	sph_neumannl(unsigned int __n, long double __x)	44
8.3	GNU Extended Mathematical Special Functions	45
8.3.1	Detailed Description	57
8.3.2	Function Documentation	57
8.3.2.1	airy_ai(_Tp __x)	57
8.3.2.2	airy_ai(std::complex< _Tp > __x)	58
8.3.2.3	airy_aif(float __x)	58
8.3.2.4	airy_ail(long double __x)	58
8.3.2.5	airy_bi(_Tp __x)	59
8.3.2.6	airy_bi(std::complex< _Tp > __x)	59
8.3.2.7	airy_bif(float __x)	59
8.3.2.8	airy_bil(long double __x)	60
8.3.2.9	bernoulli(unsigned int __n)	60

8.3.2.10	bernoulli(unsigned int __n, _Tp __x)	60
8.3.2.11	bernoullif(unsigned int __n)	61
8.3.2.12	bernoullil(unsigned int __n)	61
8.3.2.13	binomial(unsigned int __n, unsigned int __k)	61
8.3.2.14	binomial_cdf(_Tp __p, unsigned int __n, unsigned int __k)	62
8.3.2.15	binomial_pdf(_Tp __p, unsigned int __n, unsigned int __k)	62
8.3.2.16	binomialf(unsigned int __n, unsigned int __k)	63
8.3.2.17	binomiall(unsigned int __n, unsigned int __k)	63
8.3.2.18	bose_einstein(_Tps __s, _Tp __x)	63
8.3.2.19	bose_einsteinf(float __s, float __x)	63
8.3.2.20	bose_einsteinl(long double __s, long double __x)	63
8.3.2.21	chebyshev_t(unsigned int __n, _Tp __x)	63
8.3.2.22	chebyshev_tf(unsigned int __n, float __x)	64
8.3.2.23	chebyshev_tl(unsigned int __n, long double __x)	64
8.3.2.24	chebyshev_u(unsigned int __n, _Tp __x)	64
8.3.2.25	chebyshev_uf(unsigned int __n, float __x)	65
8.3.2.26	chebyshev_ul(unsigned int __n, long double __x)	65
8.3.2.27	chebyshev_v(unsigned int __n, _Tp __x)	65
8.3.2.28	chebyshev_vf(unsigned int __n, float __x)	66
8.3.2.29	chebyshev_vl(unsigned int __n, long double __x)	66
8.3.2.30	chebyshev_w(unsigned int __n, _Tp __x)	66
8.3.2.31	chebyshev_wf(unsigned int __n, float __x)	67
8.3.2.32	chebyshev_wl(unsigned int __n, long double __x)	67
8.3.2.33	clausen(unsigned int __m, _Tp __w)	67
8.3.2.34	clausen(unsigned int __m, std::complex< _Tp > __z)	68
8.3.2.35	clausen_cl(unsigned int __m, _Tp __x)	68
8.3.2.36	clausen_clf(unsigned int __m, float __x)	69
8.3.2.37	clausen_cll(unsigned int __m, long double __x)	69

8.3.2.38	<code>clausen_sl(unsigned int __m, _Tp __x)</code>	69
8.3.2.39	<code>clausen_slf(unsigned int __m, float __x)</code>	70
8.3.2.40	<code>clausen_sll(unsigned int __m, long double __x)</code>	70
8.3.2.41	<code>clausenf(unsigned int __m, float __x)</code>	70
8.3.2.42	<code>clausenf(unsigned int __m, std::complex< float > __z)</code>	71
8.3.2.43	<code>clausenl(unsigned int __m, long double __x)</code>	71
8.3.2.44	<code>clausenl(unsigned int __m, std::complex< long double > __z)</code>	71
8.3.2.45	<code>comp_ellint_d(_Tk __k)</code>	71
8.3.2.46	<code>comp_ellint_df(float __k)</code>	72
8.3.2.47	<code>comp_ellint_dl(long double __k)</code>	72
8.3.2.48	<code>comp_ellint_rf(float __x, float __y)</code>	72
8.3.2.49	<code>comp_ellint_rf(long double __x, long double __y)</code>	72
8.3.2.50	<code>comp_ellint_rf(_Tx __x, _Ty __y)</code>	72
8.3.2.51	<code>comp_ellint_rg(float __x, float __y)</code>	73
8.3.2.52	<code>comp_ellint_rg(long double __x, long double __y)</code>	73
8.3.2.53	<code>comp_ellint_rg(_Tx __x, _Ty __y)</code>	73
8.3.2.54	<code>conf_hyperg(_Tpa __a, _Tpc __c, _Tp __x)</code>	74
8.3.2.55	<code>conf_hyperg_lim(_Tpc __c, _Tp __x)</code>	74
8.3.2.56	<code>conf_hyperg_limf(float __c, float __x)</code>	75
8.3.2.57	<code>conf_hyperg_liml(long double __c, long double __x)</code>	75
8.3.2.58	<code>conf_hypergf(float __a, float __c, float __x)</code>	75
8.3.2.59	<code>conf_hypergl(long double __a, long double __c, long double __x)</code>	75
8.3.2.60	<code>cos_pi(_Tp __x)</code>	75
8.3.2.61	<code>cos_pif(float __x)</code>	76
8.3.2.62	<code>cos_pil(long double __x)</code>	76
8.3.2.63	<code>cosh_pi(_Tp __x)</code>	76
8.3.2.64	<code>cosh_pif(float __x)</code>	77
8.3.2.65	<code>cosh_pil(long double __x)</code>	77

8.3.2.66	<code>coshint(_Tp __x)</code>	77
8.3.2.67	<code>coshintf(float __x)</code>	78
8.3.2.68	<code>coshintl(long double __x)</code>	78
8.3.2.69	<code>cosint(_Tp __x)</code>	78
8.3.2.70	<code>cosintf(float __x)</code>	79
8.3.2.71	<code>cosintl(long double __x)</code>	79
8.3.2.72	<code>cyl_hankel_1(_Tpnu __nu, _Tp __z)</code>	79
8.3.2.73	<code>cyl_hankel_1(std::complex< _Tpnu > __nu, std::complex< _Tp > __x)</code>	80
8.3.2.74	<code>cyl_hankel_1f(float __nu, float __z)</code>	80
8.3.2.75	<code>cyl_hankel_1f(std::complex< float > __nu, std::complex< float > __x)</code>	80
8.3.2.76	<code>cyl_hankel_1l(long double __nu, long double __z)</code>	81
8.3.2.77	<code>cyl_hankel_1l(std::complex< long double > __nu, std::complex< long double > __x)</code>	81
8.3.2.78	<code>cyl_hankel_2(_Tpnu __nu, _Tp __z)</code>	81
8.3.2.79	<code>cyl_hankel_2(std::complex< _Tpnu > __nu, std::complex< _Tp > __x)</code>	82
8.3.2.80	<code>cyl_hankel_2f(float __nu, float __z)</code>	82
8.3.2.81	<code>cyl_hankel_2f(std::complex< float > __nu, std::complex< float > __x)</code>	82
8.3.2.82	<code>cyl_hankel_2l(long double __nu, long double __z)</code>	83
8.3.2.83	<code>cyl_hankel_2l(std::complex< long double > __nu, std::complex< long double > __x)</code>	83
8.3.2.84	<code>dawson(_Tp __x)</code>	83
8.3.2.85	<code>dawsonf(float __x)</code>	83
8.3.2.86	<code>dawsonl(long double __x)</code>	84
8.3.2.87	<code>debye(unsigned int __n, _Tp __x)</code>	84
8.3.2.88	<code>debyef(unsigned int __n, float __x)</code>	84
8.3.2.89	<code>debyel(unsigned int __n, long double __x)</code>	85
8.3.2.90	<code>dilog(_Tp __x)</code>	85
8.3.2.91	<code>dilogf(float __x)</code>	85
8.3.2.92	<code>dilogl(long double __x)</code>	85
8.3.2.93	<code>dirichlet_beta(_Tp __s)</code>	86

8.3.2.94	dirichlet_betaf(float __s)	86
8.3.2.95	dirichlet_betaf(long double __s)	86
8.3.2.96	dirichlet_eta(_Tp __s)	87
8.3.2.97	dirichlet_etaf(float __s)	87
8.3.2.98	dirichlet_etaf(long double __s)	87
8.3.2.99	dirichlet_lambda(_Tp __s)	88
8.3.2.100	dirichlet_lambdaf(float __s)	88
8.3.2.101	dirichlet_lambdaf(long double __s)	88
8.3.2.102	double_factorial(int __n)	89
8.3.2.103	double_factorialf(int __n)	89
8.3.2.104	double_factoriall(int __n)	89
8.3.2.105	ellint_cel(_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)	89
8.3.2.106	ellint_celf(float __k_c, float __p, float __a, float __b)	90
8.3.2.107	ellint_cell(long double __k_c, long double __p, long double __a, long double __b)	90
8.3.2.108	ellint_d(_Tk __k, _Tphi __phi)	90
8.3.2.109	ellint_df(float __k, float __phi)	91
8.3.2.110	ellint_dl(long double __k, long double __phi)	91
8.3.2.111	ellint_el1(_Tp __x, _Tk __k_c)	91
8.3.2.112	ellint_el1f(float __x, float __k_c)	92
8.3.2.113	ellint_el1l(long double __x, long double __k_c)	92
8.3.2.114	ellint_el2(_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)	92
8.3.2.115	ellint_el2f(float __x, float __k_c, float __a, float __b)	93
8.3.2.116	ellint_el2l(long double __x, long double __k_c, long double __a, long double __b)	93
8.3.2.117	ellint_el3(_Tx __x, _Tk __k_c, _Tp __p)	93
8.3.2.118	ellint_el3f(float __x, float __k_c, float __p)	94
8.3.2.119	ellint_el3l(long double __x, long double __k_c, long double __p)	94
8.3.2.120	ellint_rc(_Tp __x, _Up __y)	94
8.3.2.121	ellint_rcf(float __x, float __y)	95

8.3.2.122 ellint_rcl(long double __x, long double __y)	95
8.3.2.123 ellint_rd(_Tp __x, _Up __y, _Vp __z)	95
8.3.2.124 ellint_rdf(float __x, float __y, float __z)	96
8.3.2.125 ellint_rdl(long double __x, long double __y, long double __z)	96
8.3.2.126 ellint_rf(_Tp __x, _Up __y, _Vp __z)	96
8.3.2.127 ellint_rff(float __x, float __y, float __z)	97
8.3.2.128 ellint_rfl(long double __x, long double __y, long double __z)	97
8.3.2.129 ellint_rg(_Tp __x, _Up __y, _Vp __z)	97
8.3.2.130 ellint_rgf(float __x, float __y, float __z)	98
8.3.2.131 ellint_rgl(long double __x, long double __y, long double __z)	98
8.3.2.132 ellint_rj(_Tp __x, _Up __y, _Vp __z, _Wp __p)	98
8.3.2.133 ellint_rjf(float __x, float __y, float __z, float __p)	99
8.3.2.134 ellint_rjl(long double __x, long double __y, long double __z, long double __p)	99
8.3.2.135 ellnome(_Tp __k)	99
8.3.2.136 ellnomef(float __k)	100
8.3.2.137 ellnomel(long double __k)	100
8.3.2.138 euler(unsigned int __n)	100
8.3.2.139 eulerian_1(unsigned int __n, unsigned int __m)	100
8.3.2.140 eulerian_2(unsigned int __n, unsigned int __m)	101
8.3.2.141 expint(unsigned int __n, _Tp __x)	101
8.3.2.142 expintf(unsigned int __n, float __x)	101
8.3.2.143 expintl(unsigned int __n, long double __x)	102
8.3.2.144 exponential_cdf(_Tlam __lambda, _Tp __x)	102
8.3.2.145 exponential_pdf(_Tlam __lambda, _Tp __x)	102
8.3.2.146 factorial(unsigned int __n)	102
8.3.2.147 factorialf(unsigned int __n)	103
8.3.2.148 factoriall(unsigned int __n)	103
8.3.2.149 falling_factorial(_Tp __a, _Tnu __nu)	103

8.3.2.150 falling_factorialf(float __a, float __nu)	103
8.3.2.151 falling_factoriall(long double __a, long double __nu)	104
8.3.2.152 fermi_dirac(_Tp __s, _Tp __x)	104
8.3.2.153 fermi_diracf(float __s, float __x)	104
8.3.2.154 fermi_diracl(long double __s, long double __x)	104
8.3.2.155 fisher_f_cdf(_Tp __F, unsigned int __nu1, unsigned int __nu2)	104
8.3.2.156 fisher_f_pdf(_Tp __F, unsigned int __nu1, unsigned int __nu2)	105
8.3.2.157 fresnel_c(_Tp __x)	105
8.3.2.158 fresnel_cf(float __x)	105
8.3.2.159 fresnel_cl(long double __x)	105
8.3.2.160 fresnel_s(_Tp __x)	106
8.3.2.161 fresnel_sf(float __x)	106
8.3.2.162 fresnel_sl(long double __x)	106
8.3.2.163 gamma_cdf(_Ta __alpha, _Tb __beta, _Tp __x)	106
8.3.2.164 gamma_pdf(_Ta __alpha, _Tb __beta, _Tp __x)	106
8.3.2.165 gamma_reciprocal(_Ta __a)	107
8.3.2.166 gamma_reciprocalf(float __a)	107
8.3.2.167 gamma_reciprocall(long double __a)	107
8.3.2.168 gegenbauer(unsigned int __n, _Talpha __alpha, _Tp __x)	107
8.3.2.169 gegenbauerf(unsigned int __n, float __alpha, float __x)	108
8.3.2.170 gegenbauerl(unsigned int __n, long double __alpha, long double __x)	108
8.3.2.171 harmonic(unsigned int __n)	108
8.3.2.172 heuman_lambda(_Tk __k, _Tphi __phi)	109
8.3.2.173 heuman_lambdaf(float __k, float __phi)	109
8.3.2.174 heuman_lambdal(long double __k, long double __phi)	109
8.3.2.175 hurwitz_zeta(_Tp __s, _Up __a)	110
8.3.2.176 hurwitz_zeta(_Tp __s, std::complex< _Up > __a)	111
8.3.2.177 hurwitz_zetaf(float __s, float __a)	111

8.3.2.178 hurwitz_zetal(long double __s, long double __a)	111
8.3.2.179 hyperg(_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)	111
8.3.2.180 hypergf(float __a, float __b, float __c, float __x)	112
8.3.2.181 hypergl(long double __a, long double __b, long double __c, long double __x)	112
8.3.2.182 ibeta(_Ta __a, _Tb __b, _Tp __x)	112
8.3.2.183 ibetac(_Ta __a, _Tb __b, _Tp __x)	113
8.3.2.184 ibetacf(float __a, float __b, float __x)	113
8.3.2.185 ibetacl(long double __a, long double __b, long double __x)	113
8.3.2.186 ibetaf(float __a, float __b, float __x)	114
8.3.2.187 ibetal(long double __a, long double __b, long double __x)	114
8.3.2.188 jacobi(unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)	114
8.3.2.189 jacobi_cn(_Kp __k, _Up __u)	115
8.3.2.190 jacobi_cnf(float __k, float __u)	115
8.3.2.191 jacobi_cnl(long double __k, long double __u)	115
8.3.2.192 jacobi_dn(_Kp __k, _Up __u)	116
8.3.2.193 jacobi_dnf(float __k, float __u)	116
8.3.2.194 jacobi_dnl(long double __k, long double __u)	116
8.3.2.195 jacobi_sn(_Kp __k, _Up __u)	117
8.3.2.196 jacobi_snf(float __k, float __u)	117
8.3.2.197 jacobi_snl(long double __k, long double __u)	117
8.3.2.198 jacobi_zeta(_Tk __k, _Tphi __phi)	118
8.3.2.199 jacobi_zetaf(float __k, float __phi)	118
8.3.2.200 jacobi_zetal(long double __k, long double __phi)	118
8.3.2.201 jacobif(unsigned __n, float __alpha, float __beta, float __x)	118
8.3.2.202 jacobil(unsigned __n, long double __alpha, long double __beta, long double __x)	119
8.3.2.203 lbinomial(unsigned int __n, unsigned int __k)	119
8.3.2.204 lbinomialf(unsigned int __n, unsigned int __k)	119
8.3.2.205 lbinomiall(unsigned int __n, unsigned int __k)	120

8.3.2.206 ldouble_factorial(int __n)	120
8.3.2.207 ldouble_factorialf(int __n)	120
8.3.2.208 ldouble_factoriall(int __n)	120
8.3.2.209 legendre_q(unsigned int __l, _Tp __x)	120
8.3.2.210 legendre_qf(unsigned int __l, float __x)	121
8.3.2.211 legendre_ql(unsigned int __l, long double __x)	121
8.3.2.212 lfactorial(unsigned int __n)	121
8.3.2.213 lfactorialf(unsigned int __n)	122
8.3.2.214 lfactoriall(unsigned int __n)	122
8.3.2.215 lfalling_factorial(_Tp __a, _Tnu __nu)	122
8.3.2.216 lfalling_factorialf(float __a, float __nu)	122
8.3.2.217 lfalling_factoriall(long double __a, long double __nu)	123
8.3.2.218 lgamma(_Ta __a)	123
8.3.2.219 lgamma(std::complex< _Ta > __a)	123
8.3.2.220 lgammaf(float __a)	123
8.3.2.221 lgammaf(std::complex< float > __a)	123
8.3.2.222 lgammal(long double __a)	124
8.3.2.223 lgammal(std::complex< long double > __a)	124
8.3.2.224 logint(_Tp __x)	124
8.3.2.225 logintf(float __x)	124
8.3.2.226 logintl(long double __x)	125
8.3.2.227 logistic_cdf(_Ta __a, _Tb __b, _Tp __x)	125
8.3.2.228 logistic_pdf(_Ta __a, _Tb __b, _Tp __x)	125
8.3.2.229 lognormal_cdf(_Tmu __mu, _Tsig __sigma, _Tp __x)	125
8.3.2.230 lognormal_pdf(_Tmu __mu, _Tsig __sigma, _Tp __x)	126
8.3.2.231 lrising_factorial(_Tp __a, _Tnu __nu)	126
8.3.2.232 lrising_factorialf(float __a, float __nu)	126
8.3.2.233 lrising_factoriall(long double __a, long double __nu)	126

8.3.2.234 normal_cdf(_Tmu __mu, _Tsig __sigma, _Tp __x)	127
8.3.2.235 normal_pdf(_Tmu __mu, _Tsig __sigma, _Tp __x)	127
8.3.2.236 owens_t(_Tph __h, _Tpa __a)	127
8.3.2.237 owens_tf(float __h, float __a)	127
8.3.2.238 owens_tl(long double __h, long double __a)	128
8.3.2.239 pgamma(_Ta __a, _Tp __x)	128
8.3.2.240 pgammaf(float __a, float __x)	128
8.3.2.241 pgammal(long double __a, long double __x)	128
8.3.2.242 polylog(_Tp __s, _Wp __w)	128
8.3.2.243 polylog(_Tp __s, std::complex< _Tp > __w)	129
8.3.2.244 polylogf(float __s, float __w)	129
8.3.2.245 polylogf(float __s, std::complex< float > __w)	129
8.3.2.246 polylogl(long double __s, long double __w)	129
8.3.2.247 polylogl(long double __s, std::complex< long double > __w)	130
8.3.2.248 psi(_Tp __x)	130
8.3.2.249 psif(float __x)	130
8.3.2.250 psil(long double __x)	130
8.3.2.251 qgamma(_Ta __a, _Tp __x)	131
8.3.2.252 qgammaf(float __a, float __x)	131
8.3.2.253 qgammal(long double __a, long double __x)	131
8.3.2.254 radpoly(unsigned int __n, unsigned int __m, _Tp __rho)	131
8.3.2.255 radpolyf(unsigned int __n, unsigned int __m, float __rho)	132
8.3.2.256 radpolyl(unsigned int __n, unsigned int __m, long double __rho)	132
8.3.2.257 rising_factorial(_Tp __a, _Tnu __nu)	132
8.3.2.258 rising_factorialf(float __a, float __nu)	132
8.3.2.259 rising_factoriall(long double __a, long double __nu)	133
8.3.2.260 sin_pi(_Tp __x)	133
8.3.2.261 sin_pif(float __x)	133

8.3.2.262 <code>sin_pil(long double __x)</code>	133
8.3.2.263 <code>sinc(_Tp __x)</code>	134
8.3.2.264 <code>sinc_pi(_Tp __x)</code>	134
8.3.2.265 <code>sinc_pif(float __x)</code>	134
8.3.2.266 <code>sinc_pil(long double __x)</code>	135
8.3.2.267 <code>sincf(float __x)</code>	135
8.3.2.268 <code>sincl(long double __x)</code>	135
8.3.2.269 <code>sincos(double __x)</code>	135
8.3.2.270 <code>sincos(_Tp __x)</code>	136
8.3.2.271 <code>sincos_pi(_Tp __x)</code>	136
8.3.2.272 <code>sincos_pif(float __x)</code>	136
8.3.2.273 <code>sincos_pil(long double __x)</code>	136
8.3.2.274 <code>sincosf(float __x)</code>	136
8.3.2.275 <code>sincosl(long double __x)</code>	137
8.3.2.276 <code>sinh_pi(_Tp __x)</code>	137
8.3.2.277 <code>sinh_pif(float __x)</code>	137
8.3.2.278 <code>sinh_pil(long double __x)</code>	137
8.3.2.279 <code>sinhc(_Tp __x)</code>	138
8.3.2.280 <code>sinhc_pi(_Tp __x)</code>	138
8.3.2.281 <code>sinhc_pif(float __x)</code>	138
8.3.2.282 <code>sinhc_pil(long double __x)</code>	139
8.3.2.283 <code>sinhcf(float __x)</code>	139
8.3.2.284 <code>sinhcl(long double __x)</code>	139
8.3.2.285 <code>sinhint(_Tp __x)</code>	139
8.3.2.286 <code>sinhintf(float __x)</code>	140
8.3.2.287 <code>sinhintl(long double __x)</code>	140
8.3.2.288 <code>sinint(_Tp __x)</code>	140
8.3.2.289 <code>sinintf(float __x)</code>	141

8.3.2.290 <code>sinintl(long double __x)</code>	141
8.3.2.291 <code>sph_bessel_i(unsigned int __n, _Tp __x)</code>	141
8.3.2.292 <code>sph_bessel_if(unsigned int __n, float __x)</code>	142
8.3.2.293 <code>sph_bessel_il(unsigned int __n, long double __x)</code>	142
8.3.2.294 <code>sph_bessel_k(unsigned int __n, _Tp __x)</code>	142
8.3.2.295 <code>sph_bessel_kf(unsigned int __n, float __x)</code>	143
8.3.2.296 <code>sph_bessel_kl(unsigned int __n, long double __x)</code>	143
8.3.2.297 <code>sph_hankel_1(unsigned int __n, _Tp __z)</code>	143
8.3.2.298 <code>sph_hankel_1(unsigned int __n, std::complex< _Tp > __x)</code>	144
8.3.2.299 <code>sph_hankel_1f(unsigned int __n, float __z)</code>	144
8.3.2.300 <code>sph_hankel_1f(unsigned int __n, std::complex< float > __x)</code>	144
8.3.2.301 <code>sph_hankel_1l(unsigned int __n, long double __z)</code>	145
8.3.2.302 <code>sph_hankel_1l(unsigned int __n, std::complex< long double > __x)</code>	145
8.3.2.303 <code>sph_hankel_2(unsigned int __n, _Tp __z)</code>	145
8.3.2.304 <code>sph_hankel_2(unsigned int __n, std::complex< _Tp > __x)</code>	146
8.3.2.305 <code>sph_hankel_2f(unsigned int __n, float __z)</code>	146
8.3.2.306 <code>sph_hankel_2f(unsigned int __n, std::complex< float > __x)</code>	146
8.3.2.307 <code>sph_hankel_2l(unsigned int __n, long double __z)</code>	147
8.3.2.308 <code>sph_hankel_2l(unsigned int __n, std::complex< long double > __x)</code>	147
8.3.2.309 <code>sph_harmonic(unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)</code>	147
8.3.2.310 <code>sph_harmonicf(unsigned int __l, int __m, float __theta, float __phi)</code>	147
8.3.2.311 <code>sph_harmonicl(unsigned int __l, int __m, long double __theta, long double __phi)</code>	148
8.3.2.312 <code>stirling_1(unsigned int __n, unsigned int __m)</code>	148
8.3.2.313 <code>stirling_2(unsigned int __n, unsigned int __m)</code>	149
8.3.2.314 <code>student_t_cdf(_Tt __t, unsigned int __nu)</code>	149
8.3.2.315 <code>student_t_pdf(_Tt __t, unsigned int __nu)</code>	149
8.3.2.316 <code>tan_pi(_Tp __x)</code>	150
8.3.2.317 <code>tan_pif(float __x)</code>	150

8.3.2.318 tan_pil(long double __x)	150
8.3.2.319 tanh_pi(_Tp __x)	150
8.3.2.320 tanh_pif(float __x)	151
8.3.2.321 tanh_pil(long double __x)	151
8.3.2.322 tgamma(_Ta __a)	151
8.3.2.323 tgamma(std::complex< _Ta > __a)	151
8.3.2.324 tgamma(_Ta __a, _Tp __x)	152
8.3.2.325 tgamma_lower(_Ta __a, _Tp __x)	152
8.3.2.326 tgamma_lowerf(float __a, float __x)	152
8.3.2.327 tgamma_lowerl(long double __a, long double __x)	152
8.3.2.328 tgammaf(float __a)	153
8.3.2.329 tgammaf(std::complex< float > __a)	153
8.3.2.330 tgammaf(float __a, float __x)	153
8.3.2.331 tgamma_l(long double __a)	153
8.3.2.332 tgamma_l(std::complex< long double > __a)	154
8.3.2.333 tgamma_l(long double __a, long double __x)	154
8.3.2.334 theta_1(_Tpnu __nu, _Tp __x)	154
8.3.2.335 theta_1f(float __nu, float __x)	154
8.3.2.336 theta_1l(long double __nu, long double __x)	155
8.3.2.337 theta_2(_Tpnu __nu, _Tp __x)	155
8.3.2.338 theta_2f(float __nu, float __x)	155
8.3.2.339 theta_2l(long double __nu, long double __x)	155
8.3.2.340 theta_3(_Tpnu __nu, _Tp __x)	156
8.3.2.341 theta_3f(float __nu, float __x)	156
8.3.2.342 theta_3l(long double __nu, long double __x)	156
8.3.2.343 theta_4(_Tpnu __nu, _Tp __x)	156
8.3.2.344 theta_4f(float __nu, float __x)	157
8.3.2.345 theta_4l(long double __nu, long double __x)	157

8.3.2.346	<code>theta_c(_Tpk __k, _Tp __x)</code>	157
8.3.2.347	<code>theta_cf(float __k, float __x)</code>	158
8.3.2.348	<code>theta_cl(long double __k, long double __x)</code>	158
8.3.2.349	<code>theta_d(_Tpk __k, _Tp __x)</code>	158
8.3.2.350	<code>theta_df(float __k, float __x)</code>	159
8.3.2.351	<code>theta_dl(long double __k, long double __x)</code>	159
8.3.2.352	<code>theta_n(_Tpk __k, _Tp __x)</code>	159
8.3.2.353	<code>theta_nf(float __k, float __x)</code>	160
8.3.2.354	<code>theta_nl(long double __k, long double __x)</code>	160
8.3.2.355	<code>theta_s(_Tpk __k, _Tp __x)</code>	160
8.3.2.356	<code>theta_sf(float __k, float __x)</code>	161
8.3.2.357	<code>theta_sl(long double __k, long double __x)</code>	161
8.3.2.358	<code>tricomi_u(_Tpa __a, _Tpc __c, _Tp __x)</code>	161
8.3.2.359	<code>tricomi_uf(float __a, float __c, float __x)</code>	162
8.3.2.360	<code>tricomi_ul(long double __a, long double __c, long double __x)</code>	162
8.3.2.361	<code>weibull_cdf(_Ta __a, _Tb __b, _Tp __x)</code>	162
8.3.2.362	<code>weibull_pdf(_Ta __a, _Tb __b, _Tp __x)</code>	163
8.3.2.363	<code>zernike(unsigned int __n, int __m, _Trho __rho, _Tphi __phi)</code>	163
8.3.2.364	<code>zernikef(unsigned int __n, int __m, float __rho, float __phi)</code>	164
8.3.2.365	<code>zernikel(unsigned int __n, int __m, long double __rho, long double __phi)</code>	164

9 Namespace Documentation	165
9.1 <code>__gnu_cxx</code> Namespace Reference	165
9.1.1 Function Documentation	178
9.1.1.1 <code>__fp_is_equal(_Tp __a, _Tp __b, _Tp __mul=_Tp{1})</code>	178
9.1.1.2 <code>__fp_is_even_integer(_Tp __a, _Tp __mul=_Tp{1})</code>	178
9.1.1.3 <code>__fp_is_half_integer(_Tp __a, _Tp __mul=_Tp{1})</code>	179
9.1.1.4 <code>__fp_is_half_odd_integer(_Tp __a, _Tp __mul=_Tp{1})</code>	179
9.1.1.5 <code>__fp_is_integer(_Tp __a, _Tp __mul=_Tp{1})</code>	180
9.1.1.6 <code>__fp_is_odd_integer(_Tp __a, _Tp __mul=_Tp{1})</code>	180
9.1.1.7 <code>__fp_is_zero(_Tp __a, _Tp __mul=_Tp{1})</code>	181
9.1.1.8 <code>__fp_max_abs(_Tp __a, _Tp __b)</code>	181
9.1.1.9 <code>__parity(_IntTp __k)</code>	181
9.2 <code>std</code> Namespace Reference	182
9.3 <code>std::detail</code> Namespace Reference	184
9.3.1 Function Documentation	206
9.3.1.1 <code>__airy(_Tp __z)</code>	206
9.3.1.2 <code>__airy_ai(std::complex< _Tp > __z)</code>	206
9.3.1.3 <code>__airy_arg(std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)</code>	207
9.3.1.4 <code>__airy_bi(std::complex< _Tp > __z)</code>	207
9.3.1.5 <code>__assoc_laguerre(unsigned int __n, unsigned int __m, _Tp __x)</code>	207
9.3.1.6 <code>__assoc_legendre_p(unsigned int __l, unsigned int __m, _Tp __x)</code>	208
9.3.1.7 <code>__bernoulli(unsigned int __n)</code>	208
9.3.1.8 <code>__bernoulli(unsigned int __n, _Tp __x)</code>	209
9.3.1.9 <code>__bernoulli_2n(unsigned int __n)</code>	209
9.3.1.10 <code>__bernoulli_series(unsigned int __n)</code>	210
9.3.1.11 <code>__beta(_Tp __a, _Tp __b)</code>	210
9.3.1.12 <code>__beta_gamma(_Tp __a, _Tp __b)</code>	211

9.3.1.13	<code>__beta_inc(_Tp __a, _Tp __b, _Tp __x)</code>	211
9.3.1.14	<code>__beta_lgamma(_Tp __a, _Tp __b)</code>	212
9.3.1.15	<code>__beta_product(_Tp __a, _Tp __b)</code>	212
9.3.1.16	<code>__binomial(unsigned int __n, unsigned int __k)</code>	213
9.3.1.17	<code>__binomial(_Tp __nu, unsigned int __k)</code>	214
9.3.1.18	<code>__binomial_cdf(_Tp __p, unsigned int __n, unsigned int __k)</code>	214
9.3.1.19	<code>__binomial_cdfc(_Tp __p, unsigned int __n, unsigned int __k)</code>	215
9.3.1.20	<code>__binomial_pdf(_Tp __p, unsigned int __n, unsigned int __k)</code>	215
9.3.1.21	<code>__bose_einstein(_Sp __s, _Tp __x)</code>	215
9.3.1.22	<code>__chebyshev_recur(unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1)</code>	216
9.3.1.23	<code>__chebyshev_t(unsigned int __n, _Tp __x)</code>	217
9.3.1.24	<code>__chebyshev_u(unsigned int __n, _Tp __x)</code>	217
9.3.1.25	<code>__chebyshev_v(unsigned int __n, _Tp __x)</code>	218
9.3.1.26	<code>__chebyshev_w(unsigned int __n, _Tp __x)</code>	218
9.3.1.27	<code>__chi_squared_pdf(_Tp __chi2, unsigned int __nu)</code>	219
9.3.1.28	<code>__chi_squared_pdfc(_Tp __chi2, unsigned int __nu)</code>	219
9.3.1.29	<code>__chshint(_Tp __x, _Tp & _Chi, _Tp & _Shi)</code>	219
9.3.1.30	<code>__chshint_cont_frac(_Tp __t, _Tp & _Chi, _Tp & _Shi)</code>	220
9.3.1.31	<code>__chshint_series(_Tp __t, _Tp & _Chi, _Tp & _Shi)</code>	220
9.3.1.32	<code>__clamp_0_m2pi(std::complex< _Tp > __z)</code>	220
9.3.1.33	<code>__clamp_pi(std::complex< _Tp > __z)</code>	220
9.3.1.34	<code>__clausen(unsigned int __m, std::complex< _Tp > __z)</code>	220
9.3.1.35	<code>__clausen(unsigned int __m, _Tp __x)</code>	221
9.3.1.36	<code>__clausen_cl(unsigned int __m, std::complex< _Tp > __z)</code>	221
9.3.1.37	<code>__clausen_cl(unsigned int __m, _Tp __x)</code>	222
9.3.1.38	<code>__clausen_sl(unsigned int __m, std::complex< _Tp > __z)</code>	222
9.3.1.39	<code>__clausen_sl(unsigned int __m, _Tp __x)</code>	223
9.3.1.40	<code>__comp_ellint_1(_Tp __k)</code>	223

9.3.1.41	<code>__comp_ellint_2(_Tp __k)</code>	224
9.3.1.42	<code>__comp_ellint_3(_Tp __k, _Tp __nu)</code>	224
9.3.1.43	<code>__comp_ellint_d(_Tp __k)</code>	225
9.3.1.44	<code>__comp_ellint_rf(_Tp __x, _Tp __y)</code>	225
9.3.1.45	<code>__comp_ellint_rg(_Tp __x, _Tp __y)</code>	225
9.3.1.46	<code>__conf_hyperg(_Tp __a, _Tp __c, _Tp __x)</code>	225
9.3.1.47	<code>__conf_hyperg_lim(_Tp __c, _Tp __x)</code>	226
9.3.1.48	<code>__conf_hyperg_lim_series(_Tp __c, _Tp __x)</code>	226
9.3.1.49	<code>__conf_hyperg_luke(_Tp __a, _Tp __c, _Tp __xin)</code>	227
9.3.1.50	<code>__conf_hyperg_series(_Tp __a, _Tp __c, _Tp __x)</code>	227
9.3.1.51	<code>__cos_pi(_Tp __x)</code>	228
9.3.1.52	<code>__cos_pi(std::complex< _Tp > __z)</code>	228
9.3.1.53	<code>__cosh_pi(_Tp __x)</code>	228
9.3.1.54	<code>__cosh_pi(std::complex< _Tp > __z)</code>	228
9.3.1.55	<code>__coshint(const _Tp __x)</code>	228
9.3.1.56	<code>__cyl_bessel(std::complex< _Tp > __nu, std::complex< _Tp > __z)</code>	229
9.3.1.57	<code>__cyl_bessel_i(_Tp __nu, _Tp __x)</code>	229
9.3.1.58	<code>__cyl_bessel_ij_series(_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)</code>	230
9.3.1.59	<code>__cyl_bessel_ik(_Tp __nu, _Tp __x)</code>	230
9.3.1.60	<code>__cyl_bessel_ik_asymp(_Tp __nu, _Tp __x)</code>	231
9.3.1.61	<code>__cyl_bessel_ik_steel(_Tp __nu, _Tp __x)</code>	231
9.3.1.62	<code>__cyl_bessel_j(_Tp __nu, _Tp __x)</code>	232
9.3.1.63	<code>__cyl_bessel_jn(_Tp __nu, _Tp __x)</code>	232
9.3.1.64	<code>__cyl_bessel_jn_asymp(_Tp __nu, _Tp __x)</code>	232
9.3.1.65	<code>__cyl_bessel_jn_neg_arg(_Tp __nu, _Tp __x)</code>	233
9.3.1.66	<code>__cyl_bessel_jn_steel(_Tp __nu, _Tp __x)</code>	233
9.3.1.67	<code>__cyl_bessel_k(_Tp __nu, _Tp __x)</code>	234
9.3.1.68	<code>__cyl_hankel_1(_Tp __nu, _Tp __x)</code>	234

9.3.1.69	<code>__cyl_hankel_1(std::complex<_Tp> __nu, std::complex<_Tp> __z)</code>	235
9.3.1.70	<code>__cyl_hankel_2(_Tp __nu, _Tp __x)</code>	235
9.3.1.71	<code>__cyl_hankel_2(std::complex<_Tp> __nu, std::complex<_Tp> __z)</code>	235
9.3.1.72	<code>__cyl_neumann(std::complex<_Tp> __nu, std::complex<_Tp> __z)</code>	236
9.3.1.73	<code>__cyl_neumann_n(_Tp __nu, _Tp __x)</code>	236
9.3.1.74	<code>__dawson(_Tp __x)</code>	237
9.3.1.75	<code>__dawson_cont_frac(_Tp __x)</code>	237
9.3.1.76	<code>__dawson_series(_Tp __x)</code>	237
9.3.1.77	<code>__debye(unsigned int __n, _Tp __x)</code>	238
9.3.1.78	<code>__debye_region(std::complex<_Tp> __alpha, int &__indexr, char &__aorb)</code>	238
9.3.1.79	<code>__dilog(_Tp __x)</code>	239
9.3.1.80	<code>__dirichlet_beta(std::complex<_Tp> __s)</code>	239
9.3.1.81	<code>__dirichlet_beta(_Tp __s)</code>	240
9.3.1.82	<code>__dirichlet_eta(std::complex<_Tp> __s)</code>	240
9.3.1.83	<code>__dirichlet_eta(_Tp __s)</code>	241
9.3.1.84	<code>__dirichlet_lambda(_Tp __s)</code>	241
9.3.1.85	<code>__double_factorial(int __n)</code>	241
9.3.1.86	<code>__ellint_1(_Tp __k, _Tp __phi)</code>	242
9.3.1.87	<code>__ellint_2(_Tp __k, _Tp __phi)</code>	242
9.3.1.88	<code>__ellint_3(_Tp __k, _Tp __nu, _Tp __phi)</code>	243
9.3.1.89	<code>__ellint_cel(_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)</code>	243
9.3.1.90	<code>__ellint_d(_Tp __k, _Tp __phi)</code>	244
9.3.1.91	<code>__ellint_el1(_Tp __x, _Tp __k_c)</code>	244
9.3.1.92	<code>__ellint_el2(_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)</code>	244
9.3.1.93	<code>__ellint_el3(_Tp __x, _Tp __k_c, _Tp __p)</code>	244
9.3.1.94	<code>__ellint_rc(_Tp __x, _Tp __y)</code>	244
9.3.1.95	<code>__ellint_rd(_Tp __x, _Tp __y, _Tp __z)</code>	245
9.3.1.96	<code>__ellint_rf(_Tp __x, _Tp __y, _Tp __z)</code>	246

9.3.1.97	<code>__ellint_rg(_Tp __x, _Tp __y, _Tp __z)</code>	246
9.3.1.98	<code>__ellint_rj(_Tp __x, _Tp __y, _Tp __z, _Tp __p)</code>	247
9.3.1.99	<code>__ellnome(_Tp __k)</code>	248
9.3.1.100	<code>__ellnome_k(_Tp __k)</code>	248
9.3.1.101	<code>__ellnome_series(_Tp __k)</code>	248
9.3.1.102	<code>__euler(unsigned int __n)</code>	248
9.3.1.103	<code>__euler(unsigned int __n, _Tp __x)</code>	249
9.3.1.104	<code>__euler_series(unsigned int __n)</code>	249
9.3.1.105	<code>__eulerian_1_recur(unsigned int __n, unsigned int __m)</code>	249
9.3.1.106	<code>__eulerian_2(unsigned int __n, unsigned int __m)</code>	250
9.3.1.107	<code>__eulerian_2_recur(unsigned int __n, unsigned int __m)</code>	250
9.3.1.108	<code>__expint(unsigned int __n, _Tp __x)</code>	250
9.3.1.109	<code>__expint(_Tp __x)</code>	251
9.3.1.110	<code>__expint_E1(_Tp __x)</code>	251
9.3.1.111	<code>__expint_E1_asymp(_Tp __x)</code>	252
9.3.1.112	<code>__expint_E1_series(_Tp __x)</code>	252
9.3.1.113	<code>__expint_Ei(_Tp __x)</code>	252
9.3.1.114	<code>__expint_Ei_asymp(_Tp __x)</code>	253
9.3.1.115	<code>__expint_Ei_series(_Tp __x)</code>	253
9.3.1.116	<code>__expint_En_asymp(unsigned int __n, _Tp __x)</code>	254
9.3.1.117	<code>__expint_En_cont_frac(unsigned int __n, _Tp __x)</code>	254
9.3.1.118	<code>__expint_En_large_n(unsigned int __n, _Tp __x)</code>	255
9.3.1.119	<code>__expint_En_recursion(unsigned int __n, _Tp __x)</code>	255
9.3.1.120	<code>__expint_En_series(unsigned int __n, _Tp __x)</code>	256
9.3.1.121	<code>__exponential_cdf(_Tp __lambda, _Tp __x)</code>	257
9.3.1.122	<code>__exponential_cdfc(_Tp __lambda, _Tp __x)</code>	257
9.3.1.123	<code>__exponential_pdf(_Tp __lambda, _Tp __x)</code>	257
9.3.1.124	<code>__factorial(unsigned int __n)</code>	257

9.3.1.125	<code>__falling_factorial(_Tp __a, int __n)</code>	258
9.3.1.126	<code>__falling_factorial(_Tp __a, _Tp __nu)</code>	258
9.3.1.127	<code>__fermi_dirac(_Sp __s, _Tp __x)</code>	258
9.3.1.128	<code>__fisher_f_cdf(_Tp __F, unsigned int __nu1, unsigned int __nu2)</code>	259
9.3.1.129	<code>__fisher_f_cdfc(_Tp __F, unsigned int __nu1, unsigned int __nu2)</code>	259
9.3.1.130	<code>__fisher_f_pdf(_Tp __F, unsigned int __nu1, unsigned int __nu2)</code>	260
9.3.1.131	<code>__fock_airy(_Tp __x)</code>	260
9.3.1.132	<code>__fresnel(const _Tp __x)</code>	260
9.3.1.133	<code>__fresnel_cont_frac(const _Tp __ax, _Tp &_Cf, _Tp &_Sf)</code>	261
9.3.1.134	<code>__fresnel_series(const _Tp __ax, _Tp &_Cf, _Tp &_Sf)</code>	261
9.3.1.135	<code>__gamma(_Tp __a)</code>	261
9.3.1.136	<code>__gamma(_Tp __a, _Tp __x)</code>	262
9.3.1.137	<code>__gamma_cdf(_Tp __alpha, _Tp __beta, _Tp __x)</code>	262
9.3.1.138	<code>__gamma_cdfc(_Tp __alpha, _Tp __beta, _Tp __x)</code>	262
9.3.1.139	<code>__gamma_cont_frac(_Tp __a, _Tp __x)</code>	263
9.3.1.140	<code>__gamma_pdf(_Tp __alpha, _Tp __beta, _Tp __x)</code>	263
9.3.1.141	<code>__gamma_reciprocal(_Tp __a)</code>	263
9.3.1.142	<code>__gamma_reciprocal_series(_Tp __a)</code>	264
9.3.1.143	<code>__gamma_series(_Tp __a, _Tp __x)</code>	264
9.3.1.144	<code>__gamma_temme(_Tp __mu)</code>	265
9.3.1.145	<code>__gauss(_Tp __x)</code>	266
9.3.1.146	<code>__gegenbauer_poly(unsigned int __n, _Tp __alpha, _Tp __x)</code>	266
9.3.1.147	<code>__hankel(std::complex<_Tp> __nu, std::complex<_Tp> __z)</code>	267
9.3.1.148	<code>__hankel_debye(std::complex<_Tp> __nu, std::complex<_Tp> __z, std::complex<_Tp> __alpha, int __indx, char &__aorb, int &__morn)</code>	267
9.3.1.149	<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>	267

9.3.1.150	<code>__hankel_uniform(std::complex< _Tp > __nu, std::complex< _Tp > __z)</code>	268
9.3.1.151	<code>__hankel_uniform_olover(std::complex< _Tp > __nu, std::complex< _Tp > __z)</code>	268
9.3.1.152	<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>	269
9.3.1.153	<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>	269
9.3.1.154	<code>__harmonic_number(unsigned int __n)</code>	270
9.3.1.155	<code>__hermite_zeros(unsigned int __n, __Tp __proto=__Tp{})</code>	270
9.3.1.156	<code>__heuman_lambda(__Tp __k, __Tp __phi)</code>	270
9.3.1.157	<code>__hurwitz_zeta(__Tp __s, __Tp __a)</code>	270
9.3.1.158	<code>__hurwitz_zeta_euler_maclaurin(__Tp __s, __Tp __a)</code>	271
9.3.1.159	<code>__hurwitz_zeta_polylog(__Tp __s, std::complex< _Tp > __a)</code>	271
9.3.1.160	<code>__hydrogen(unsigned int __n, unsigned int __l, unsigned int __m, __Tp __Z, __Tp __r, __Tp __theta, __Tp __phi)</code>	272
9.3.1.161	<code>__hyperg(__Tp __a, __Tp __b, __Tp __c, __Tp __x)</code>	272
9.3.1.162	<code>__hyperg_luke(__Tp __a, __Tp __b, __Tp __c, __Tp __xin)</code>	272
9.3.1.163	<code>__hyperg_reflect(__Tp __a, __Tp __b, __Tp __c, __Tp __x)</code>	273
9.3.1.164	<code>__hyperg_series(__Tp __a, __Tp __b, __Tp __c, __Tp __x)</code>	273
9.3.1.165	<code>__ibeta_cont_frac(__Tp __a, __Tp __b, __Tp __x)</code>	274
9.3.1.166	<code>__jacobi_sncndn(__Tp __k, __Tp __u)</code>	274
9.3.1.167	<code>__jacobi_zeta(__Tp __k, __Tp __phi)</code>	274
9.3.1.168	<code>__laguerre(unsigned int __n, __Tp __x)</code>	274
9.3.1.169	<code>__laguerre_zeros(unsigned int __n, __Tp __alpha)</code>	275
9.3.1.170	<code>__lanczos_binet1p(__Tp __z)</code>	275

9.3.1.171	<code>__lanczos_log_gamma1p(_Tp __z)</code>	276
9.3.1.172	<code>__legendre_q(unsigned int __l, _Tp __x)</code>	276
9.3.1.173	<code>__legendre_zeros(unsigned int __l, _Tp proto=_Tp{})</code>	276
9.3.1.174	<code>__log_binomial(unsigned int __n, unsigned int __k)</code>	277
9.3.1.175	<code>__log_binomial(_Tp __nu, unsigned int __k)</code>	277
9.3.1.176	<code>__log_binomial_sign(_Tp __nu, unsigned int __k)</code>	278
9.3.1.177	<code>__log_binomial_sign(std::complex< _Tp > __nu, unsigned int __k)</code>	278
9.3.1.178	<code>__log_double_factorial(_Tp __x)</code>	278
9.3.1.179	<code>__log_double_factorial(int __n)</code>	279
9.3.1.180	<code>__log_factorial(unsigned int __n)</code>	279
9.3.1.181	<code>__log_falling_factorial(_Tp __a, _Tp __nu)</code>	279
9.3.1.182	<code>__log_gamma(_Tp __a)</code>	279
9.3.1.183	<code>__log_gamma(std::complex< _Tp > __a)</code>	280
9.3.1.184	<code>__log_gamma_bernoulli(_Tp __x)</code>	280
9.3.1.185	<code>__log_gamma_sign(_Tp __a)</code>	281
9.3.1.186	<code>__log_gamma_sign(std::complex< _Tp > __a)</code>	281
9.3.1.187	<code>__log_rising_factorial(_Tp __a, _Tp __nu)</code>	281
9.3.1.188	<code>__log_stirling_1(unsigned int __n, unsigned int __m)</code>	282
9.3.1.189	<code>__log_stirling_1_sign(unsigned int __n, unsigned int __m)</code>	282
9.3.1.190	<code>__log_stirling_2(unsigned int __n, unsigned int __m)</code>	282
9.3.1.191	<code>__logint(const _Tp __x)</code>	282
9.3.1.192	<code>__logistic_cdf(_Tp __a, _Tp __b, _Tp __x)</code>	283
9.3.1.193	<code>__logistic_pdf(_Tp __a, _Tp __b, _Tp __x)</code>	283
9.3.1.194	<code>__lognormal_cdf(_Tp __mu, _Tp __sigma, _Tp __x)</code>	283
9.3.1.195	<code>__lognormal_pdf(_Tp __nu, _Tp __sigma, _Tp __x)</code>	283
9.3.1.196	<code>__normal_cdf(_Tp __mu, _Tp __sigma, _Tp __x)</code>	284
9.3.1.197	<code>__normal_pdf(_Tp __mu, _Tp __sigma, _Tp __x)</code>	284
9.3.1.198	<code>__owens_t(_Tp __h, _Tp __a)</code>	284

9.3.1.199	<code>__pgamma(_Tp __a, _Tp __x)</code>	285
9.3.1.200	<code>__polar_pi(_Tp __rho, _Tp __phi_pi)</code>	285
9.3.1.201	<code>__poly_hermite(unsigned int __n, _Tp __x)</code>	285
9.3.1.202	<code>__poly_hermite_asymp(unsigned int __n, _Tp __x)</code>	286
9.3.1.203	<code>__poly_hermite_recursion(unsigned int __n, _Tp __x)</code>	287
9.3.1.204	<code>__poly_jacobi(unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)</code>	287
9.3.1.205	<code>__poly_laguerre(unsigned int __n, _Tpa __alpha1, _Tp __x)</code>	287
9.3.1.206	<code>__poly_laguerre_hyperg(unsigned int __n, _Tpa __alpha1, _Tp __x)</code>	288
9.3.1.207	<code>__poly_laguerre_large_n(unsigned __n, _Tpa __alpha1, _Tp __x)</code>	289
9.3.1.208	<code>__poly_laguerre_recursion(unsigned int __n, _Tpa __alpha1, _Tp __x)</code>	289
9.3.1.209	<code>__poly_legendre_p(unsigned int __l, _Tp __x)</code>	290
9.3.1.210	<code>__poly_prob_hermite_recursion(unsigned int __n, _Tp __x)</code>	291
9.3.1.211	<code>__poly_radial_jacobi(unsigned int __n, unsigned int __m, _Tp __rho)</code>	291
9.3.1.212	<code>__polylog(_Tp __s, _Tp __x)</code>	292
9.3.1.213	<code>__polylog(_Tp __s, std::complex<_Tp> __w)</code>	293
9.3.1.214	<code>__polylog_exp(_Tp __s, _ArgType __w)</code>	293
9.3.1.215	<code>__polylog_exp_asymp(_Tp __s, std::complex<_Tp> __w)</code>	294
9.3.1.216	<code>__polylog_exp_neg(_Tp __s, std::complex<_Tp> __w)</code>	294
9.3.1.217	<code>__polylog_exp_neg(int __n, std::complex<_Tp> __w)</code>	295
9.3.1.218	<code>__polylog_exp_neg_int(int __s, std::complex<_Tp> __w)</code>	295
9.3.1.219	<code>__polylog_exp_neg_int(int __s, _Tp __w)</code>	296
9.3.1.220	<code>__polylog_exp_neg_real(_Tp __s, std::complex<_Tp> __w)</code>	296
9.3.1.221	<code>__polylog_exp_neg_real(_Tp __s, _Tp __w)</code>	297
9.3.1.222	<code>__polylog_exp_pos(unsigned int __s, std::complex<_Tp> __w)</code>	297
9.3.1.223	<code>__polylog_exp_pos(unsigned int __s, _Tp __w)</code>	298
9.3.1.224	<code>__polylog_exp_pos(_Tp __s, std::complex<_Tp> __w)</code>	299
9.3.1.225	<code>__polylog_exp_pos_int(unsigned int __s, std::complex<_Tp> __w)</code>	299
9.3.1.226	<code>__polylog_exp_pos_int(unsigned int __s, _Tp __w)</code>	300

9.3.1.227	<code>__polylog_exp_pos_real(_Tp __s, std::complex< _Tp > __w)</code>	300
9.3.1.228	<code>__polylog_exp_pos_real(_Tp __s, _Tp __w)</code>	301
9.3.1.229	<code>__polylog_exp_sum(_PowTp __s, _Tp __w)</code>	301
9.3.1.230	<code>__psi(unsigned int __n)</code>	302
9.3.1.231	<code>__psi(_Tp __x)</code>	302
9.3.1.232	<code>__psi(unsigned int __n, _Tp __x)</code>	302
9.3.1.233	<code>__psi_asymp(_Tp __x)</code>	303
9.3.1.234	<code>__psi_series(_Tp __x)</code>	303
9.3.1.235	<code>__qgamma(_Tp __a, _Tp __x)</code>	303
9.3.1.236	<code>__rice_pdf(_Tp __nu, _Tp __sigma, _Tp __x)</code>	304
9.3.1.237	<code>__riemann_zeta(_Tp __s)</code>	304
9.3.1.238	<code>__riemann_zeta_euler_maclaurin(_Tp __s)</code>	304
9.3.1.239	<code>__riemann_zeta_glob(_Tp __s)</code>	305
9.3.1.240	<code>__riemann_zeta_m_1(_Tp __s)</code>	305
9.3.1.241	<code>__riemann_zeta_m_1_glob(_Tp __s)</code>	305
9.3.1.242	<code>__riemann_zeta_product(_Tp __s)</code>	306
9.3.1.243	<code>__riemann_zeta_sum(_Tp __s)</code>	306
9.3.1.244	<code>__rising_factorial(_Tp __a, int __n)</code>	307
9.3.1.245	<code>__rising_factorial(_Tp __a, _Tp __nu)</code>	307
9.3.1.246	<code>__sin_pi(_Tp __x)</code>	307
9.3.1.247	<code>__sin_pi(std::complex< _Tp > __z)</code>	308
9.3.1.248	<code>__sinc(_Tp __x)</code>	308
9.3.1.249	<code>__sinc_pi(_Tp __x)</code>	308
9.3.1.250	<code>__sincos(_Tp __x)</code>	308
9.3.1.251	<code>__sincos(float __x)</code>	308
9.3.1.252	<code>__sincos(double __x)</code>	308
9.3.1.253	<code>__sincos(long double __x)</code>	309
9.3.1.254	<code>__sincos_pi(_Tp __x)</code>	309

9.3.1.255	<code>__sincosint(_Tp __x)</code>	309
9.3.1.256	<code>__sincosint_asymp(_Tp __t, _Tp &_Si, _Tp &_Ci)</code>	309
9.3.1.257	<code>__sincosint_cont_frac(_Tp __t, _Tp &_Si, _Tp &_Ci)</code>	309
9.3.1.258	<code>__sincosint_series(_Tp __t, _Tp &_Si, _Tp &_Ci)</code>	310
9.3.1.259	<code>__sinh_pi(_Tp __x)</code>	310
9.3.1.260	<code>__sinh_pi(std::complex< _Tp > __z)</code>	310
9.3.1.261	<code>__sinhc(_Tp __x)</code>	310
9.3.1.262	<code>__sinhc_pi(_Tp __x)</code>	310
9.3.1.263	<code>__sinhint(const _Tp __x)</code>	310
9.3.1.264	<code>__sph_bessel(unsigned int __n, _Tp __x)</code>	311
9.3.1.265	<code>__sph_bessel(unsigned int __n, std::complex< _Tp > __z)</code>	311
9.3.1.266	<code>__sph_bessel_ik(unsigned int __n, _Tp __x)</code>	312
9.3.1.267	<code>__sph_bessel_jn(unsigned int __n, _Tp __x)</code>	312
9.3.1.268	<code>__sph_bessel_jn_neg_arg(unsigned int __n, _Tp __x)</code>	313
9.3.1.269	<code>__sph_hankel(unsigned int __n, std::complex< _Tp > __z)</code>	313
9.3.1.270	<code>__sph_hankel_1(unsigned int __n, _Tp __x)</code>	313
9.3.1.271	<code>__sph_hankel_1(unsigned int __n, std::complex< _Tp > __z)</code>	314
9.3.1.272	<code>__sph_hankel_2(unsigned int __n, _Tp __x)</code>	314
9.3.1.273	<code>__sph_hankel_2(unsigned int __n, std::complex< _Tp > __z)</code>	315
9.3.1.274	<code>__sph_harmonic(unsigned int __l, int __m, _Tp __theta, _Tp __phi)</code>	315
9.3.1.275	<code>__sph_legendre(unsigned int __l, unsigned int __m, _Tp __theta)</code>	316
9.3.1.276	<code>__sph_neumann(unsigned int __n, _Tp __x)</code>	316
9.3.1.277	<code>__sph_neumann(unsigned int __n, std::complex< _Tp > __z)</code>	317
9.3.1.278	<code>__spouge_binet1p(_Tp __z)</code>	317
9.3.1.279	<code>__spouge_log_gamma1p(_Tp __z)</code>	318
9.3.1.280	<code>__stirling_1(unsigned int __n, unsigned int __m)</code>	319
9.3.1.281	<code>__stirling_1_recur(unsigned int __n, unsigned int __m)</code>	319
9.3.1.282	<code>__stirling_1_series(unsigned int __n, unsigned int __m)</code>	319

9.3.1.283	<code>__stirling_2(unsigned int __n, unsigned int __m)</code>	320
9.3.1.284	<code>__stirling_2_recur(unsigned int __n, unsigned int __m)</code>	320
9.3.1.285	<code>__stirling_2_series(unsigned int __n, unsigned int __m)</code>	320
9.3.1.286	<code>__student_t_cdf(_Tp __t, unsigned int __nu)</code>	320
9.3.1.287	<code>__student_t_cdfc(_Tp __t, unsigned int __nu)</code>	321
9.3.1.288	<code>__student_t_pdf(_Tp __t, unsigned int __nu)</code>	321
9.3.1.289	<code>__tan_pi(_Tp __x)</code>	322
9.3.1.290	<code>__tan_pi(std::complex< _Tp > __z)</code>	322
9.3.1.291	<code>__tanh_pi(_Tp __x)</code>	322
9.3.1.292	<code>__tanh_pi(std::complex< _Tp > __z)</code>	322
9.3.1.293	<code>__tgamma(_Tp __a, _Tp __x)</code>	323
9.3.1.294	<code>__tgamma_lower(_Tp __a, _Tp __x)</code>	323
9.3.1.295	<code>__theta_1(_Tp __nu, _Tp __x)</code>	323
9.3.1.296	<code>__theta_2(_Tp __nu, _Tp __x)</code>	324
9.3.1.297	<code>__theta_2_asymp(_Tp __nu, _Tp __x)</code>	324
9.3.1.298	<code>__theta_2_sum(_Tp __nu, _Tp __x)</code>	324
9.3.1.299	<code>__theta_3(_Tp __nu, _Tp __x)</code>	324
9.3.1.300	<code>__theta_3_asymp(_Tp __nu, _Tp __x)</code>	325
9.3.1.301	<code>__theta_3_sum(_Tp __nu, _Tp __x)</code>	325
9.3.1.302	<code>__theta_4(_Tp __nu, _Tp __x)</code>	325
9.3.1.303	<code>__theta_c(_Tp __k, _Tp __x)</code>	326
9.3.1.304	<code>__theta_d(_Tp __k, _Tp __x)</code>	326
9.3.1.305	<code>__theta_n(_Tp __k, _Tp __x)</code>	326
9.3.1.306	<code>__theta_s(_Tp __k, _Tp __x)</code>	326
9.3.1.307	<code>__tricomi_u(_Tp __a, _Tp __c, _Tp __x)</code>	326
9.3.1.308	<code>__tricomi_u_naive(_Tp __a, _Tp __c, _Tp __x)</code>	327
9.3.1.309	<code>__weibull_cdf(_Tp __a, _Tp __b, _Tp __x)</code>	328
9.3.1.310	<code>__weibull_pdf(_Tp __a, _Tp __b, _Tp __x)</code>	328

9.3.1.311	<code>__zernike(unsigned int __n, int __m, _Tp __rho, _Tp __phi)</code>	328
9.3.1.312	<code>__znorm1(_Tp __x)</code>	329
9.3.1.313	<code>__znorm2(_Tp __x)</code>	329
9.3.2	Variable Documentation	329
9.3.2.1	<code>__max_FGH</code>	329
9.3.2.2	<code>__max_FGH< double ></code>	329
9.3.2.3	<code>__max_FGH< float ></code>	329
9.3.2.4	<code>_Num_Euler_Maclaurin_zeta</code>	330
9.3.2.5	<code>_S_double_factorial_table</code>	330
9.3.2.6	<code>_S_Euler_Maclaurin_zeta</code>	330
9.3.2.7	<code>_S_factorial_table</code>	330
9.3.2.8	<code>_S_harmonic_denom</code>	330
9.3.2.9	<code>_S_harmonic_numer</code>	330
9.3.2.10	<code>_S_neg_double_factorial_table</code>	331
9.3.2.11	<code>_S_num_double_factorials</code>	331
9.3.2.12	<code>_S_num_double_factorials< double ></code>	331
9.3.2.13	<code>_S_num_double_factorials< float ></code>	331
9.3.2.14	<code>_S_num_double_factorials< long double ></code>	331
9.3.2.15	<code>_S_num_factorials</code>	331
9.3.2.16	<code>_S_num_factorials< double ></code>	331
9.3.2.17	<code>_S_num_factorials< float ></code>	331
9.3.2.18	<code>_S_num_factorials< long double ></code>	331
9.3.2.19	<code>_S_num_harmonic_numer</code>	332
9.3.2.20	<code>_S_num_neg_double_factorials</code>	332
9.3.2.21	<code>_S_num_neg_double_factorials< double ></code>	332
9.3.2.22	<code>_S_num_neg_double_factorials< float ></code>	332
9.3.2.23	<code>_S_num_neg_double_factorials< long double ></code>	332
9.3.2.24	<code>_S_num_zetam1</code>	332
9.3.2.25	<code>_S_zetam1</code>	332

10 Class Documentation	333
10.1 <code>__gnu_cxx::__airy_t<_Tx, _Tp></code> Struct Template Reference	333
10.1.1 Detailed Description	333
10.1.2 Member Function Documentation	334
10.1.2.1 <code>__Wronskian() const</code>	334
10.1.3 Member Data Documentation	334
10.1.3.1 <code>__Ai_deriv</code>	334
10.1.3.2 <code>__Ai_value</code>	334
10.1.3.3 <code>__Bi_deriv</code>	334
10.1.3.4 <code>__Bi_value</code>	334
10.1.3.5 <code>__x_arg</code>	334
10.2 <code>__gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp></code> Struct Template Reference	335
10.2.1 Detailed Description	335
10.2.2 Member Function Documentation	335
10.2.2.1 <code>__Wronskian() const</code>	335
10.2.3 Member Data Documentation	336
10.2.3.1 <code>__J_deriv</code>	336
10.2.3.2 <code>__J_value</code>	336
10.2.3.3 <code>__N_deriv</code>	336
10.2.3.4 <code>__N_value</code>	336
10.2.3.5 <code>__nu_arg</code>	336
10.2.3.6 <code>__x_arg</code>	336
10.3 <code>__gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp></code> Struct Template Reference	337
10.3.1 Detailed Description	337
10.3.2 Member Function Documentation	337
10.3.2.1 <code>__Wronskian() const</code>	337
10.3.3 Member Data Documentation	338
10.3.3.1 <code>__H1_deriv</code>	338

10.3.3.2	__H1_value	338
10.3.3.3	__H2_deriv	338
10.3.3.4	__H2_value	338
10.3.3.5	__nu_arg	338
10.3.3.6	__x_arg	338
10.4	__gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp> Struct Template Reference	339
10.4.1	Detailed Description	339
10.4.2	Member Function Documentation	339
10.4.2.1	__Wronskian() const	339
10.4.3	Member Data Documentation	340
10.4.3.1	__I_deriv	340
10.4.3.2	__I_value	340
10.4.3.3	__K_deriv	340
10.4.3.4	__K_value	340
10.4.3.5	__nu_arg	340
10.4.3.6	__x_arg	340
10.5	__gnu_cxx::__fock_airy_t<_Tx, _Tp> Struct Template Reference	341
10.5.1	Detailed Description	341
10.5.2	Member Function Documentation	341
10.5.2.1	__Wronskian() const	341
10.5.3	Member Data Documentation	342
10.5.3.1	__w1_deriv	342
10.5.3.2	__w1_value	342
10.5.3.3	__w2_deriv	342
10.5.3.4	__w2_value	342
10.5.3.5	__x_arg	342
10.6	__gnu_cxx::__fp_is_integer_t Struct Reference	342
10.6.1	Detailed Description	343

10.6.2	Member Function Documentation	343
10.6.2.1	operator bool() const	343
10.6.2.2	operator>() const	343
10.6.3	Member Data Documentation	343
10.6.3.1	__is_integral	343
10.6.3.2	__value	344
10.7	__gnu_cxx::__gamma_inc_t< _Tp > Struct Template Reference	344
10.7.1	Detailed Description	344
10.7.2	Member Data Documentation	344
10.7.2.1	__lgamma_value	344
10.7.2.2	__tgamma_value	345
10.8	__gnu_cxx::__gamma_temme_t< _Tp > Struct Template Reference	345
10.8.1	Detailed Description	345
10.8.2	Member Data Documentation	346
10.8.2.1	__gamma_1_value	346
10.8.2.2	__gamma_2_value	346
10.8.2.3	__gamma_minus_value	346
10.8.2.4	__gamma_plus_value	346
10.8.2.5	__mu_arg	346
10.9	__gnu_cxx::__jacobi_t< _Tp > Struct Template Reference	346
10.9.1	Detailed Description	347
10.9.2	Member Function Documentation	347
10.9.2.1	__am() const	347
10.9.2.2	__cd() const	347
10.9.2.3	__cs() const	347
10.9.2.4	__dc() const	348
10.9.2.5	__ds() const	348
10.9.2.6	__nc() const	348

10.9.2.7	__nd() const	348
10.9.2.8	__ns() const	348
10.9.2.9	__sc() const	348
10.9.2.10	__sd() const	348
10.9.3	Member Data Documentation	348
10.9.3.1	__cn_value	348
10.9.3.2	__dn_value	349
10.9.3.3	__sn_value	349
10.10	__gnu_cxx::__lgamma_t< _Tp > Struct Template Reference	349
10.10.1	Detailed Description	349
10.10.2	Member Data Documentation	349
10.10.2.1	__lgamma_sign	349
10.10.2.2	__lgamma_value	350
10.11	__gnu_cxx::__pgamma_t< _Tp > Struct Template Reference	350
10.11.1	Detailed Description	350
10.11.2	Member Data Documentation	350
10.11.2.1	__pgamma_value	350
10.11.2.2	__qgamma_value	350
10.12	__gnu_cxx::__quadrature_point_t< _Tp > Struct Template Reference	351
10.12.1	Detailed Description	351
10.12.2	Constructor & Destructor Documentation	351
10.12.2.1	__quadrature_point_t()=default	351
10.12.2.2	__quadrature_point_t(_Tp __z, _Tp __w)	351
10.12.3	Member Data Documentation	351
10.12.3.1	__weight	351
10.12.3.2	__zero	352
10.13	__gnu_cxx::__sincos_t< _Tp > Struct Template Reference	352
10.13.1	Detailed Description	352

10.13.2 Member Data Documentation	352
10.13.2.1 __cos_v	352
10.13.2.2 __sin_v	352
10.14 __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp > Struct Template Reference	353
10.14.1 Detailed Description	353
10.14.2 Member Function Documentation	353
10.14.2.1 __Wronskian() const	353
10.14.3 Member Data Documentation	354
10.14.3.1 __j_deriv	354
10.14.3.2 __j_value	354
10.14.3.3 __n_arg	354
10.14.3.4 __n_deriv	354
10.14.3.5 __n_value	354
10.14.3.6 __x_arg	354
10.15 __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp > Struct Template Reference	355
10.15.1 Detailed Description	355
10.15.2 Member Function Documentation	355
10.15.2.1 __Wronskian() const	355
10.15.3 Member Data Documentation	356
10.15.3.1 __h1_deriv	356
10.15.3.2 __h1_value	356
10.15.3.3 __h2_deriv	356
10.15.3.4 __h2_value	356
10.15.3.5 __n_arg	356
10.15.3.6 __x_arg	356
10.16 __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp > Struct Template Reference	357
10.16.1 Detailed Description	357
10.16.2 Member Function Documentation	357

10.16.2.1 <code>__Wronskian() const</code>	357
10.16.3 Member Data Documentation	358
10.16.3.1 <code>__i_deriv</code>	358
10.16.3.2 <code>__i_value</code>	358
10.16.3.3 <code>__k_deriv</code>	358
10.16.3.4 <code>__k_value</code>	358
10.16.3.5 <code>__x_arg</code>	358
10.16.3.6 <code>n_arg</code>	358
10.17 <code>std::__detail::__gamma_lanczos_data< _Tp ></code> Struct Template Reference	359
10.17.1 Detailed Description	359
10.18 <code>std::__detail::__gamma_lanczos_data< double ></code> Struct Template Reference	359
10.18.1 Detailed Description	359
10.18.2 Member Data Documentation	359
10.18.2.1 <code>_S_cheby</code>	359
10.18.2.2 <code>_S_g</code>	360
10.19 <code>std::__detail::__gamma_lanczos_data< float ></code> Struct Template Reference	360
10.19.1 Detailed Description	360
10.19.2 Member Data Documentation	360
10.19.2.1 <code>_S_cheby</code>	360
10.19.2.2 <code>_S_g</code>	360
10.20 <code>std::__detail::__gamma_lanczos_data< long double ></code> Struct Template Reference	361
10.20.1 Detailed Description	361
10.20.2 Member Data Documentation	361
10.20.2.1 <code>_S_cheby</code>	361
10.20.2.2 <code>_S_g</code>	361
10.21 <code>std::__detail::__gamma_spouge_data< _Tp ></code> Struct Template Reference	362
10.21.1 Detailed Description	362
10.22 <code>std::__detail::__gamma_spouge_data< double ></code> Struct Template Reference	362

10.22.1 Detailed Description	362
10.22.2 Member Data Documentation	362
10.22.2.1 <code>_S_cheby</code>	362
10.23 <code>std::__detail::__gamma_spouge_data< float ></code> Struct Template Reference	363
10.23.1 Detailed Description	363
10.23.2 Member Data Documentation	363
10.23.2.1 <code>_S_cheby</code>	363
10.24 <code>std::__detail::__gamma_spouge_data< long double ></code> Struct Template Reference	363
10.24.1 Detailed Description	363
10.24.2 Member Data Documentation	364
10.24.2.1 <code>_S_cheby</code>	364
10.25 <code>std::__detail::_Airy< _Tp ></code> Class Template Reference	364
10.25.1 Detailed Description	365
10.25.2 Member Typedef Documentation	365
10.25.2.1 <code>scalar_type</code>	365
10.25.2.2 <code>value_type</code>	365
10.25.3 Constructor & Destructor Documentation	365
10.25.3.1 <code>_Airy()=default</code>	365
10.25.3.2 <code>_Airy(const _Airy &)=default</code>	365
10.25.3.3 <code>_Airy(_Airy &&)=default</code>	365
10.25.4 Member Function Documentation	365
10.25.4.1 <code>operator()(value_type __y) const</code>	365
10.25.5 Member Data Documentation	366
10.25.5.1 <code>inner_radius</code>	366
10.25.5.2 <code>outer_radius</code>	366
10.26 <code>std::__detail::_Airy_asymp< _Tp ></code> Class Template Reference	366
10.26.1 Detailed Description	367
10.26.2 Member Typedef Documentation	368

10.26.2.1 <code>_Cmplx</code>	368
10.26.3 Constructor & Destructor Documentation	368
10.26.3.1 <code>_Airy_asymp()</code> =default	368
10.26.4 Member Function Documentation	368
10.26.4.1 <code>_S_absarg_ge_pio3(_Cmplx __z) const</code>	368
10.26.4.2 <code>_S_absarg_lt_pio3(_Cmplx __z) const</code>	368
10.26.4.3 <code>operator()(_Cmplx __t, bool __return_fock_airy=false) const</code>	369
10.27std::__detail::_Airy_asymp_data< <code>_Tp</code> > Struct Template Reference	370
10.27.1 Detailed Description	370
10.28std::__detail::_Airy_asymp_data< <code>double</code> > Struct Template Reference	370
10.28.1 Detailed Description	371
10.28.2 Member Data Documentation	371
10.28.2.1 <code>_S_c</code>	371
10.28.2.2 <code>_S_d</code>	371
10.28.2.3 <code>_S_max_cd</code>	371
10.29std::__detail::_Airy_asymp_data< <code>float</code> > Struct Template Reference	371
10.29.1 Detailed Description	371
10.29.2 Member Data Documentation	372
10.29.2.1 <code>_S_c</code>	372
10.29.2.2 <code>_S_d</code>	372
10.29.2.3 <code>_S_max_cd</code>	372
10.30std::__detail::_Airy_asymp_data< <code>long double</code> > Struct Template Reference	372
10.30.1 Detailed Description	372
10.30.2 Member Data Documentation	372
10.30.2.1 <code>_S_c</code>	372
10.30.2.2 <code>_S_d</code>	373
10.30.2.3 <code>_S_max_cd</code>	373
10.31std::__detail::_Airy_asymp_series< <code>_Sum</code> > Class Template Reference	373

10.31.1 Detailed Description	373
10.31.2 Member Typedef Documentation	374
10.31.2.1 scalar_type	374
10.31.2.2 value_type	374
10.31.3 Constructor & Destructor Documentation	374
10.31.3.1 _Airy_asymp_series(_Sum __proto)	374
10.31.3.2 _Airy_asymp_series(const _Airy_asymp_series &)=default	374
10.31.3.3 _Airy_asymp_series(_Airy_asymp_series &&)=default	374
10.31.4 Member Function Documentation	374
10.31.4.1 operator()(value_type __y)	374
10.31.5 Member Data Documentation	375
10.31.5.1 _S_sqrt_pi	375
10.32std::__detail::_Airy_default_radii< _Tp > Struct Template Reference	375
10.32.1 Detailed Description	375
10.33std::__detail::_Airy_default_radii< double > Struct Template Reference	375
10.33.1 Detailed Description	375
10.33.2 Member Data Documentation	376
10.33.2.1 inner_radius	376
10.33.2.2 outer_radius	376
10.34std::__detail::_Airy_default_radii< float > Struct Template Reference	376
10.34.1 Detailed Description	376
10.34.2 Member Data Documentation	376
10.34.2.1 inner_radius	376
10.34.2.2 outer_radius	376
10.35std::__detail::_Airy_default_radii< long double > Struct Template Reference	377
10.35.1 Detailed Description	377
10.35.2 Member Data Documentation	377
10.35.2.1 inner_radius	377

10.35.2.2 outer_radius	377
10.36std::__detail::_Airy_series< _Tp > Class Template Reference	377
10.36.1 Detailed Description	378
10.36.2 Member Typedef Documentation	378
10.36.2.1 _Cmplx	378
10.36.3 Member Function Documentation	379
10.36.3.1 _S_Ai(_Cmplx __t)	379
10.36.3.2 _S_Airy(_Cmplx __t)	379
10.36.3.3 _S_Bi(_Cmplx __t)	380
10.36.3.4 _S_FGH(_Cmplx __t)	380
10.36.3.5 _S_Fock(_Cmplx __t)	380
10.36.3.6 _S_Scorer(_Cmplx __t)	381
10.36.3.7 _S_Scorer2(_Cmplx __t)	382
10.36.4 Member Data Documentation	382
10.36.4.1 _N_FGH	382
10.36.4.2 _S_Ai0	382
10.36.4.3 _S_Aip0	382
10.36.4.4 _S_Bi0	382
10.36.4.5 _S_Bip0	383
10.36.4.6 _S_eps	383
10.36.4.7 _S_Gi0	383
10.36.4.8 _S_Gip0	383
10.36.4.9 _S_Hi0	383
10.36.4.10_S_Hip0	383
10.36.4.11_S_i	383
10.36.4.12_S_pi	383
10.36.4.13_S_sqrt_pi	384
10.37std::__detail::_AiryAuxilliaryState< _Tp > Struct Template Reference	384

10.37.1 Detailed Description	384
10.37.2 Member Typedef Documentation	384
10.37.2.1 _Val	384
10.37.3 Member Data Documentation	385
10.37.3.1 __fai_deriv	385
10.37.3.2 __fai_value	385
10.37.3.3 __gai_deriv	385
10.37.3.4 __gai_value	385
10.37.3.5 __hai_deriv	385
10.37.3.6 __hai_value	385
10.37.3.7 __z	385
10.38std::__detail::_AiryState< _Tp > Struct Template Reference	385
10.38.1 Detailed Description	386
10.38.2 Member Typedef Documentation	386
10.38.2.1 _Real	386
10.38.3 Member Function Documentation	386
10.38.3.1 true_Wronskian()	386
10.38.3.2 Wronskian() const	386
10.38.4 Member Data Documentation	387
10.38.4.1 __Ai_deriv	387
10.38.4.2 __Ai_value	387
10.38.4.3 __Bi_deriv	387
10.38.4.4 __Bi_value	387
10.38.4.5 __z	387
10.39std::__detail::_AsympTerminator< _Tp > Class Template Reference	387
10.39.1 Detailed Description	388
10.39.2 Constructor & Destructor Documentation	388
10.39.2.1 _AsympTerminator(std::size_t __max_iter, _Real __mul=_Real{1})	388

10.39.3 Member Function Documentation	388
10.39.3.1 operator()(_Tp __term, _Tp __sum)	388
10.40std::__detail::_Factorial_table< _Tp > Struct Template Reference	388
10.40.1 Detailed Description	388
10.40.2 Member Data Documentation	389
10.40.2.1 __factorial	389
10.40.2.2 __log_factorial	389
10.40.2.3 __n	389
10.41std::__detail::_Terminator< _Tp > Class Template Reference	389
10.41.1 Detailed Description	389
10.41.2 Constructor & Destructor Documentation	390
10.41.2.1 _Terminator(std::size_t __max_iter, _Real __mul=_Real{1})	390
10.41.3 Member Function Documentation	390
10.41.3.1 operator()(_Tp __term, _Tp __sum)	390
11 File Documentation	391
11.1 bits/sf_airy.tcc File Reference	391
11.1.1 Detailed Description	393
11.1.2 Macro Definition Documentation	393
11.1.2.1 _GLIBCXX_BITS_SF_AIRY_TCC	393
11.2 bits/sf_bernoulli.tcc File Reference	393
11.2.1 Detailed Description	394
11.2.2 Macro Definition Documentation	394
11.2.2.1 _GLIBCXX_BITS_SF_BERNOULLI_TCC	394
11.3 bits/sf_bessel.tcc File Reference	394
11.3.1 Detailed Description	396
11.3.2 Macro Definition Documentation	397
11.3.2.1 _GLIBCXX_BITS_SF_BESSEL_TCC	397

11.4	bits/sf_beta.tcc File Reference	397
11.4.1	Detailed Description	398
11.4.2	Macro Definition Documentation	398
11.4.2.1	_GLIBCXX_BITS_SF_BETA_TCC	398
11.5	bits/sf_cardinal.tcc File Reference	399
11.5.1	Macro Definition Documentation	400
11.5.1.1	_GLIBCXX_BITS_SF_CARDINAL_TCC	400
11.6	bits/sf_chebyshev.tcc File Reference	400
11.6.1	Detailed Description	401
11.6.2	Macro Definition Documentation	402
11.6.2.1	_GLIBCXX_BITS_SF_CHEBYSHEV_TCC	402
11.7	bits/sf_dawson.tcc File Reference	402
11.7.1	Detailed Description	403
11.7.2	Macro Definition Documentation	403
11.7.2.1	_GLIBCXX_BITS_SF_DAWSON_TCC	403
11.8	bits/sf_distributions.tcc File Reference	403
11.8.1	Detailed Description	406
11.8.2	Macro Definition Documentation	406
11.8.2.1	_GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC	406
11.9	bits/sf_ellint.tcc File Reference	406
11.9.1	Detailed Description	408
11.9.2	Macro Definition Documentation	408
11.9.2.1	_GLIBCXX_BITS_SF_ELLINT_TCC	408
11.10	bits/sf_euler.tcc File Reference	409
11.10.1	Detailed Description	410
11.10.2	Macro Definition Documentation	410
11.10.2.1	_GLIBCXX_BITS_SF_EULER_TCC	410
11.11	bits/sf_expint.tcc File Reference	410

11.11.1 Detailed Description	412
11.11.2 Macro Definition Documentation	412
11.11.2.1 _GLIBCXX_BITS_SF_EXPINT_TCC	412
11.12bits/sf_fresnel.tcc File Reference	413
11.12.1 Detailed Description	414
11.12.2 Macro Definition Documentation	414
11.12.2.1 _GLIBCXX_BITS_SF_FRESNEL_TCC	414
11.13bits/sf_gamma.tcc File Reference	414
11.13.1 Detailed Description	421
11.13.2 Macro Definition Documentation	422
11.13.2.1 _GLIBCXX_BITS_SF_GAMMA_TCC	422
11.14bits/sf_gegenbauer.tcc File Reference	422
11.14.1 Detailed Description	423
11.14.2 Macro Definition Documentation	423
11.14.2.1 _GLIBCXX_BITS_SF_GEGENBAUER_TCC	423
11.15bits/sf_hankel.tcc File Reference	423
11.15.1 Detailed Description	426
11.15.2 Macro Definition Documentation	426
11.15.2.1 _GLIBCXX_BITS_SF_HANKEL_TCC	426
11.16bits/sf_hermite.tcc File Reference	426
11.16.1 Detailed Description	427
11.16.2 Macro Definition Documentation	428
11.16.2.1 _GLIBCXX_BITS_SF_HERMITE_TCC	428
11.17bits/sf_hydrogen.tcc File Reference	428
11.17.1 Detailed Description	429
11.17.2 Macro Definition Documentation	429
11.17.2.1 _GLIBCXX_BITS_SF_HYDROGEN_TCC	429
11.18bits/sf_hyperg.tcc File Reference	429

11.18.1 Detailed Description	431
11.18.2 Macro Definition Documentation	431
11.18.2.1 _GLIBCXX_BITS_SF_HYPERG_TCC	431
11.19bits/sf_hypint.tcc File Reference	431
11.19.1 Detailed Description	433
11.19.2 Macro Definition Documentation	433
11.19.2.1 _GLIBCXX_BITS_SF_HYPINT_TCC	433
11.20bits/sf_jacobi.tcc File Reference	433
11.20.1 Detailed Description	434
11.20.2 Macro Definition Documentation	434
11.20.2.1 _GLIBCXX_BITS_SF_JACOBI_TCC	434
11.21bits/sf_laguerre.tcc File Reference	435
11.21.1 Detailed Description	436
11.21.2 Macro Definition Documentation	436
11.21.2.1 _GLIBCXX_BITS_SF_LAGUERRE_TCC	436
11.22bits/sf_legendre.tcc File Reference	436
11.22.1 Detailed Description	438
11.22.2 Macro Definition Documentation	438
11.22.2.1 _GLIBCXX_BITS_SF_LEGENDRE_TCC	438
11.23bits/sf_mod_bessel.tcc File Reference	438
11.23.1 Detailed Description	440
11.23.2 Macro Definition Documentation	440
11.23.2.1 _GLIBCXX_BITS_SF_MOD_BESSEL_TCC	440
11.24bits/sf_owens_t.tcc File Reference	440
11.24.1 Detailed Description	441
11.24.2 Macro Definition Documentation	441
11.24.2.1 _GLIBCXX_BITS_SF_OWENS_T_TCC	441
11.25bits/sf_polylog.tcc File Reference	441

11.25.1 Detailed Description	444
11.25.2 Macro Definition Documentation	444
11.25.2.1 _GLIBCXX_BITS_SF_POLYLOG_TCC	444
11.26bits/sf_stirling.tcc File Reference	444
11.26.1 Detailed Description	445
11.26.2 Macro Definition Documentation	445
11.26.2.1 _GLIBCXX_BITS_SF_STIRLING_TCC	445
11.27bits/sf_theta.tcc File Reference	446
11.27.1 Detailed Description	447
11.27.2 Macro Definition Documentation	447
11.27.2.1 _GLIBCXX_BITS_SF_THETA_TCC	447
11.28bits/sf_trig.tcc File Reference	448
11.28.1 Detailed Description	449
11.28.2 Macro Definition Documentation	449
11.28.2.1 _GLIBCXX_BITS_SF_TRIG_TCC	449
11.29bits/sf_trigint.tcc File Reference	450
11.29.1 Detailed Description	451
11.29.2 Macro Definition Documentation	451
11.29.2.1 _GLIBCXX_BITS_SF_TRIGINT_TCC	451
11.30bits/sf_zeta.tcc File Reference	451
11.30.1 Detailed Description	453
11.30.2 Macro Definition Documentation	453
11.30.2.1 _GLIBCXX_BITS_SF_ZETA_TCC	453
11.31bits/specfun.h File Reference	454
11.31.1 Detailed Description	469
11.31.2 Macro Definition Documentation	469
11.31.2.1 __cpp_lib_math_special_functions	469
11.31.2.2 __STDCPP_MATH_SPEC_FUNCS__	469
11.32bits/specfun_state.h File Reference	469
11.32.1 Detailed Description	471
11.33ext/math_util.h File Reference	471
11.33.1 Detailed Description	471

Chapter 1

Mathematical Special Functions

1.1 Introduction and History

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

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

Follow-up proposals for new special functions have also been published: [A proposal to add special mathematical functions according to the ISO/IEC 80000-2:2009 standard](#), Vincent Reverdý.

[A Proposal to add Mathematical Functions for Statistics to the C++ Standard Library](#), Paul A Bristow.

[A proposal to add sincos to the standard library](#), Paul Dreik.

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

1.2 Contents

The following functions are implemented in namespace `std`:

- [assoc_laguerre](#) - Associated Laguerre functions
- [assoc_legendre](#) - Associated Legendre functions
- [beta](#) - Beta functions
- [comp_ellint_1](#) - Complete elliptic functions of the first kind
- [comp_ellint_2](#) - Complete elliptic functions of the second kind

- [comp_ellint_3](#) - Complete elliptic functions of the third kind
- [cyl_bessel_i](#) - Regular modified cylindrical Bessel functions
- [cyl_bessel_j](#) - Cylindrical Bessel functions of the first kind
- [cyl_bessel_k](#) - Irregular modified cylindrical Bessel functions
- [cyl_neumann](#) - Cylindrical Neumann functions or Cylindrical Bessel functions of the second kind
- [ellint_1](#) - Incomplete elliptic functions of the first kind
- [ellint_2](#) - Incomplete elliptic functions of the second kind
- [ellint_3](#) - Incomplete elliptic functions of the third kind
- [expint](#) - The exponential integral
- [hermite](#) - Hermite polynomials
- [laguerre](#) - Laguerre functions
- [legendre](#) - Legendre polynomials
- [riemann_zeta](#) - The Riemann zeta function
- [sph_bessel](#) - Spherical Bessel functions
- [sph_legendre](#) - Spherical Legendre functions
- [sph_neumann](#) - Spherical Neumann functions

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

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

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

- [airy_ai](#) - Airy functions of the first kind
- [airy_bi](#) - Airy functions of the second kind
- [bernoulli](#) - Bernoulli polynomials
- [binomial](#) - Binomial coefficients
- [bose_einstein](#) - Bose-Einstein integrals
- [chebyshev_t](#) - Chebyshev polynomials of the first kind
- [chebyshev_u](#) - Chebyshev polynomials of the second kind
- [chebyshev_v](#) - Chebyshev polynomials of the third kind
- [chebyshev_w](#) - Chebyshev polynomials of the fourth kind
- [clausen](#) - Clausen integrals

- [clausen_cl](#) - Clausen cosine integrals
- [clausen_sl](#) - Clausen sine integrals
- [comp_ellint_d](#) - Incomplete Legendre D elliptic integral
- [conf_hyperg_lim](#) - Confluent hypergeometric limit functions
- [cos_pi](#) - Reperiodized cosine function.
- [cosh_pi](#) - Reperiodized hyperbolic cosine function.
- [coshint](#) - Hyperbolic cosine integral
- [cosint](#) - Cosine integral
- [cyl_hankel_1](#) - Cylindrical Hankel functions of the first kind
- [cyl_hankel_2](#) - Cylindrical Hankel functions of the second kind
- [dawson](#) - Dawson integrals
- [debye](#) - Debye functions
- [dilog](#) - Dilogarithm functions
- [dirichlet_beta](#) - Dirichlet beta function
- [dirichlet_eta](#) - Dirichlet beta function
- [dirichlet_lambda](#) - Dirichlet lambda function
- [double_factorial](#) - Double factorials
- [ellint_d](#) - Legendre D elliptic integrals
- [ellint_rc](#) - Carlson elliptic functions R_C
- [ellint_rd](#) - Carlson elliptic functions R_D
- [ellint_rf](#) - Carlson elliptic functions R_F
- [ellint_rg](#) - Carlson elliptic functions R_G
- [ellint_rj](#) - Carlson elliptic functions R_J
- [ellnome](#) - Elliptic nome
- [euler](#) - Euler numbers
- [euler](#) - Euler polynomials
- [eulerian_1](#) - Eulerian numbers of the first kind
- [eulerian_2](#) - Eulerian numbers of the second kind
- [expint](#) - Exponential integrals
- [factorial](#) - Factorials
- [falling_factorial](#) - Falling factorials
- [fermi_dirac](#) - Fermi-Dirac integrals
- [fresnel_c](#) - Fresnel cosine integrals

- [fresnel_s](#) - Fresnel sine integrals
- [gamma_reciprocal](#) - Reciprocal gamma function
- [gegenbauer](#) - Gegenbauer polynomials
- [heuman_lambda](#) - Heuman lambda functions
- [hurwitz_zeta](#) - Hurwitz zeta functions
- [ibeta](#) - Regularized incomplete beta functions
- [jacobi](#) - Jacobi polynomials
- [jacobi_sn](#) - Jacobi sine amplitude functions
- [jacobi_cn](#) - Jacobi cosine amplitude functions
- [jacobi_dn](#) - Jacobi delta amplitude functions
- [jacobi_zeta](#) - Jacobi zeta functions
- [lbinomial](#) - Log binomial coefficients
- [ldouble_factorial](#) - Log double factorials
- [legendre_q](#) - Legendre functions of the second kind
- [lfactorial](#) - Log factorials
- [lfalling_factorial](#) - Log falling factorials
- [lgamma](#) - Log gamma for complex arguments
- [lrising_factorial](#) - Log rising factorials
- [owens_t](#) - Owens T functions
- [pgamma](#) - Regularized lower incomplete gamma functions
- [psi](#) - Psi or digamma function
- [qgamma](#) - Regularized upper incomplete gamma functions
- [radpoly](#) - Radial polynomials
- [rising_factorial](#) - Rising factorials
- [sinhc](#) - Hyperbolic sinus cardinal function
- [sinhc_pi](#) - Reperiodized hyperbolic sinus cardinal function
- [sinc](#) - Normalized sinus cardinal function
- [sincos](#) - Sine + cosine function
- [sincos_pi](#) - Reperiodized sine + cosine function
- [sin_pi](#) - Reperiodized sine function.
- [sinh_pi](#) - Reperiodized hyperbolic sine function.
- [sinc_pi](#) - Sinus cardinal function
- [sinhint](#) - Hyperbolic sine integral

- [sinint](#) - Sine integral
- [sph_bessel_i](#) - Spherical regular modified Bessel functions
- [sph_bessel_k](#) - Spherical 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
- [stirling_1](#) - Stirling numbers of the first kind
- [stirling_2](#) - Stirling numbers of the second kind
- [tan_pi](#) - Reperiodized tangent function.
- [tanh_pi](#) - Reperiodized hyperbolic tangent function.
- [tgamma](#) - Gamma for complex arguments
- [tgamma](#) - Upper incomplete gamma functions
- [tgamma_lower](#) - Lower incomplete gamma functions
- [theta_1](#) - Exponential theta function 1
- [theta_2](#) - Exponential theta function 2
- [theta_3](#) - Exponential theta function 3
- [theta_4](#) - Exponential theta function 4
- [tricoli_u](#) - Tricoli confluent hypergeometric function
- [zernike](#) - Zernike polynomials

1.3 General Features

1.3.1 Argument Promotion

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

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

1.3.2 NaN Arguments

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

1.4 Implementation

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

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

1.5 Testing

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

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

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

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

1.6 General Bibliography

See also

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

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

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

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

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

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

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

Chapter 2

Todo List

Member [__gnu_cxx::eulerian_1](#) (unsigned int __n, unsigned int __m)

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

Member [__gnu_cxx::eulerian_2](#) (unsigned int __n, unsigned int __m)

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

Member [__gnu_cxx::stirling_1](#) (unsigned int __n, unsigned int __m)

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

Member [__gnu_cxx::stirling_2](#) (unsigned int __n, unsigned int __m)

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

page [Mathematical Special Functions](#)

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

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

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

Member [std::__detail::__euler_series](#) (unsigned int __n)

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

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

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

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

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

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

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

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

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

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

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

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

Look into asymptotic solutions.

Member [std::__detail::__riemann_zeta](#) (_Tp __s)

Global double sum or MacLaurin series in [riemann_zeta](#)?

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

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

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

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

Find asymptotic solutions for Stirling numbers of the second kind.

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

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

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

Member `std::__detail::Airy_asymp<_Tp>::S_absarg_lt_pio3` (_Cmplx __z) const

Revisit these numbers of terms for the Airy asymptotic expansions.

Member `std::__detail::Airy_series<_Tp>::S_Scorer` (_Cmplx __t)

Find out what is wrong with the $H_i = f_{ai} + g_{ai} + h_{ai}$ 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	45

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

__gnu_cxx	165
std	182
std::__detail	184

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

<code>gnu_cxx::__airy_t<_Tx, _Tp></code>	333
<code>gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp></code>	335
<code>gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp></code>	337
<code>gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp></code>	339
<code>gnu_cxx::__fock_airy_t<_Tx, _Tp></code>	341
<code>gnu_cxx::__fp_is_integer_t</code>	342
<code>gnu_cxx::__gamma_inc_t<_Tp></code>	344
<code>gnu_cxx::__gamma_temme_t<_Tp></code>	345
<code>gnu_cxx::__jacobi_t<_Tp></code>	346
<code>gnu_cxx::__lgamma_t<_Tp></code>	349
<code>gnu_cxx::__pqgamma_t<_Tp></code>	350
<code>gnu_cxx::__quadrature_point_t<_Tp></code>	351
<code>gnu_cxx::__sincos_t<_Tp></code>	352
<code>gnu_cxx::__sph_bessel_t<_Tn, _Tx, _Tp></code>	353
<code>gnu_cxx::__sph_hankel_t<_Tn, _Tx, _Tp></code>	355
<code>gnu_cxx::__sph_mod_bessel_t<_Tn, _Tx, _Tp></code>	357
<code>std::__detail::__gamma_lanczos_data<_Tp></code>	359
<code>std::__detail::__gamma_lanczos_data<double></code>	359
<code>std::__detail::__gamma_lanczos_data<float></code>	360
<code>std::__detail::__gamma_lanczos_data<long double></code>	361
<code>std::__detail::__gamma_spouge_data<_Tp></code>	362
<code>std::__detail::__gamma_spouge_data<double></code>	362
<code>std::__detail::__gamma_spouge_data<float></code>	363
<code>std::__detail::__gamma_spouge_data<long double></code>	363
<code>std::__detail::__Airy<_Tp></code>	364
<code>std::__detail::__Airy_asymp_data<_Tp></code>	370
<code>std::__detail::__Airy_asymp<_Tp></code>	366
<code>std::__detail::__Airy_asymp_data<double></code>	370
<code>std::__detail::__Airy_asymp_data<float></code>	371
<code>std::__detail::__Airy_asymp_data<long double></code>	372
<code>std::__detail::__Airy_asymp_series<_Sum></code>	373

std::__detail::_Airy_default_radii<_Tp>	375
std::__detail::_Airy_default_radii< double >	375
std::__detail::_Airy_default_radii< float >	376
std::__detail::_Airy_default_radii< long double >	377
std::__detail::_Airy_series<_Tp>	377
std::__detail::_AiryAuxilliaryState<_Tp>	384
std::__detail::_AiryState<_Tp>	385
std::__detail::_AsympTerminator<_Tp>	387
std::__detail::_Factorial_table<_Tp>	388
std::__detail::_Terminator<_Tp>	389

Chapter 6

Class Index

6.1 Class List

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

__gnu_cxx::__airy_t<_Tx, _Tp>	333
__gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp>	335
__gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp>	337
__gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp>	339
__gnu_cxx::__fock_airy_t<_Tx, _Tp>	341
__gnu_cxx::__fp_is_integer_t	342
__gnu_cxx::__gamma_inc_t<_Tp>	344
__gnu_cxx::__gamma_temme_t<_Tp>	

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

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

and

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

where $-1/2 \leq \mu \leq 1/2$ 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

__gnu_cxx::__jacobi_t<_Tp>	345
__gnu_cxx::__lgamma_t<_Tp>	346
__gnu_cxx::__pqgamma_t<_Tp>	349
__gnu_cxx::__quadrature_point_t<_Tp>	350
__gnu_cxx::__sincos_t<_Tp>	351
__gnu_cxx::__sph_bessel_t<_Tn, _Tx, _Tp>	352
__gnu_cxx::__sph_hankel_t<_Tn, _Tx, _Tp>	353
__gnu_cxx::__sph_mod_bessel_t<_Tn, _Tx, _Tp>	355
__gnu_cxx::__sph_mod_bessel_t<_Tn, _Tx, _Tp>	357
std::__detail::__gamma_lanczos_data<_Tp>	359
std::__detail::__gamma_lanczos_data<double>	359
std::__detail::__gamma_lanczos_data<float>	360
std::__detail::__gamma_lanczos_data<long double>	361
std::__detail::__gamma_spouge_data<_Tp>	362

std::__detail::__gamma_spouge_data< double >	362
std::__detail::__gamma_spouge_data< float >	363
std::__detail::__gamma_spouge_data< long double >	363
std::__detail::__Airy< _Tp >	364
std::__detail::__Airy_asymp< _Tp >	366
std::__detail::__Airy_asymp_data< _Tp >	370
std::__detail::__Airy_asymp_data< double >	370
std::__detail::__Airy_asymp_data< float >	371
std::__detail::__Airy_asymp_data< long double >	372
std::__detail::__Airy_asymp_series< _Sum >	373
std::__detail::__Airy_default_radii< _Tp >	375
std::__detail::__Airy_default_radii< double >	375
std::__detail::__Airy_default_radii< float >	376
std::__detail::__Airy_default_radii< long double >	377
std::__detail::__Airy_series< _Tp >	377
std::__detail::__AiryAuxilliaryState< _Tp >	384
std::__detail::__AiryState< _Tp >	385
std::__detail::__AsympTerminator< _Tp >	387
std::__detail::__Factorial_table< _Tp >	388
std::__detail::__Terminator< _Tp >	389

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

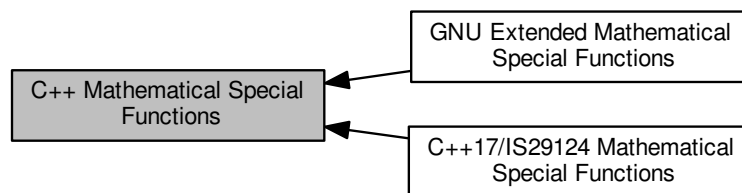
bits/sf_airy.tcc	391
bits/sf_bernoulli.tcc	393
bits/sf_bessel.tcc	394
bits/sf_beta.tcc	397
bits/sf_cardinal.tcc	399
bits/sf_chebyshev.tcc	400
bits/sf_dawson.tcc	402
bits/sf_distributions.tcc	403
bits/sf_ellint.tcc	406
bits/sf_euler.tcc	409
bits/sf_expint.tcc	410
bits/sf_fresnel.tcc	413
bits/sf_gamma.tcc	414
bits/sf_gegenbauer.tcc	422
bits/sf_hankel.tcc	423
bits/sf_hermite.tcc	426
bits/sf_hydrogen.tcc	428
bits/sf_hyperg.tcc	429
bits/sf_hypint.tcc	431
bits/sf_jacobi.tcc	433
bits/sf_laguerre.tcc	435
bits/sf_legendre.tcc	436
bits/sf_mod_bessel.tcc	438
bits/sf_owens_t.tcc	440
bits/sf_polylog.tcc	441
bits/sf_stirling.tcc	444
bits/sf_theta.tcc	446
bits/sf_trig.tcc	448
bits/sf_trigint.tcc	450
bits/sf_zeta.tcc	451
bits/specfun.h	454
bits/specfun_state.h	469
ext/math_util.h	471

Chapter 8

Module Documentation

8.1 C++ Mathematical Special Functions

Collaboration diagram for C++ Mathematical Special Functions:



Modules

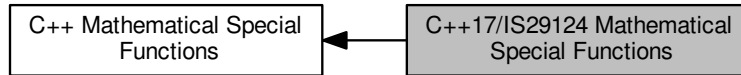
- [C++17/IS29124 Mathematical Special Functions](#)
- [GNU Extended Mathematical Special Functions](#)

8.1.1 Detailed Description

A collection of advanced mathematical special functions.

8.2 C++17/IS29124 Mathematical Special Functions

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



Functions

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::assoc_laguerre` (unsigned int __n, unsigned int __m, _Tp __x)
- `float std::assoc_laguerref` (unsigned int __n, unsigned int __m, float __x)
- `long double std::assoc_laguerrel` (unsigned int __n, unsigned int __m, long double __x)
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::assoc_legendre` (unsigned int __l, unsigned int __m, _Tp __x)
- `float std::assoc_legendref` (unsigned int __l, unsigned int __m, float __x)
- `long double std::assoc_legendrel` (unsigned int __l, unsigned int __m, long double __x)
- `template<typename _Tpa, typename _Tpb >`
`__gnu_cxx::__promote_fp_t< _Tpa, _Tpb > std::beta` (_Tpa __a, _Tpb __b)
- `float std::betaf` (float __a, float __b)
- `long double std::betal` (long double __a, long double __b)
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::comp_ellint_1` (_Tp __k)
- `float std::comp_ellint_1f` (float __k)
- `long double std::comp_ellint_1l` (long double __k)
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::comp_ellint_2` (_Tp __k)
- `float std::comp_ellint_2f` (float __k)
- `long double std::comp_ellint_2l` (long double __k)
- `template<typename _Tp, typename _Tpn >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tpn > std::comp_ellint_3` (_Tp __k, _Tpn __nu)
- `float std::comp_ellint_3f` (float __k, float __nu)
- *Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for float modulus k .*
- `long double std::comp_ellint_3l` (long double __k, long double __nu)
- *Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for long double modulus k .*
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > std::cyl_bessel_i` (_Tpnu __nu, _Tp __x)
- `float std::cyl_bessel_if` (float __nu, float __x)
- `long double std::cyl_bessel_il` (long double __nu, long double __x)
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > std::cyl_bessel_j` (_Tpnu __nu, _Tp __x)
- `float std::cyl_bessel_jf` (float __nu, float __x)
- `long double std::cyl_bessel_jl` (long double __nu, long double __x)

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

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

8.2.1 Detailed Description

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

8.2.2 Function Documentation

8.2.2.1 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x) [inline]`

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

The associated Laguerre function of real degree α , $L_n^\alpha(x)$, is defined by

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

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

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

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

where the Laguerre polynomial is defined by:

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

and $x \geq 0$.

See also

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 415 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 367 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 378 of file specfun.h.

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

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

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

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

See also

[legendre](#) for details of the Legendre function of degree 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 463 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 430 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 441 of file `specfun.h`.

8.2.2.7 `template<typename _Tpa, typename _Tpb> __gnu_cxx::__promote_fp_t<_Tpa, _Tpb> std::beta (_Tpa __a, _Tpb __b) [inline]`

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

The beta function is defined by

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

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

Template Parameters

<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 508 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 477 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 487 of file `specfun.h`.

8.2.2.10 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::comp_ellint_1 (_Tp __k) [inline]`

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

The complete elliptic integral of the first kind is defined as

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

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

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 556 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 523 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 533 of file `specfun.h`.

8.2.2.13 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::comp_ellint_2 (_Tp __k) [inline]`

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

The complete elliptic integral of the second kind is defined as

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

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 603 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 571 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 581 of file `specfun.h`.

8.2.2.16 `template<typename _Tp, typename _Tpn> __gnu_cxx::__promote_fp_t<_Tp, _Tpn> std::comp_ellint_3 (_Tp __k, _Tpn __nu) [inline]`

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

The complete elliptic integral of the third kind is defined as

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

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

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>_Tpn</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 654 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 618 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 628 of file `specfun.h`.

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

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

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 700 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 669 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 679 of file `specfun.h`.

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

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

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 746 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 715 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 725 of file `specfun.h`.

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

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

The irregular modified Bessel function is defined by:

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

where for integral $\nu = n$ a limit is taken: $\lim_{\nu \rightarrow n}$. 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 798 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 761 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 771 of file `specfun.h`.

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

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

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 846 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 813 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 823 of file `specfun.h`.

8.2.2.31 `template<typename _Tp, typename _Tpp> __gnu_cxx::__promote_fp_t<_Tp, _Tpp> std::ellint_1 (_Tp __k, _Tpp __phi) [inline]`

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

The incomplete elliptic integral of the first kind is defined as

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

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

See also

[comp_ellint_1](#).

Template Parameters

<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 894 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 861 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 871 of file `specfun.h`.

8.2.2.34 `template<typename _Tp, typename _Tpp> __gnu_cxx::__promote_fp_t<_Tp, _Tpp> std::ellint_2 (_Tp __k, _Tpp __phi) [inline]`

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

The incomplete elliptic integral of the second kind is defined as

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

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 942 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 909 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 919 of file `specfun.h`.

8.2.2.37 `template<typename _Tp, typename _Tpn, typename _Tpp> __gnu_cxx::__promote_fp_t<_Tp, _Tpn, _Tpp> std::ellint_3 (_Tp __k, _Tpn __nu, _Tpp __phi) [inline]`

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

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

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

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

See also

[comp_ellint_3](#).

Template Parameters

<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 995 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 957 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 967 of file specfun.h.

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

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

The exponential integral is given by

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

Template Parameters

<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 1035 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 1009 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 1019 of file specfun.h.

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

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

The Hermite polynomial is defined by:

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

The Hermite polynomial obeys a reflection formula:

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

Template Parameters

<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 1083 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 1050 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 1060 of file `specfun.h`.

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

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

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 >= 0</code>

Exceptions

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

Definition at line 1127 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 1098 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 1108 of file `specfun.h`.

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

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

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 1172 of file `specfun.h`.

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

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

See also

[legendre](#) for more details.

Definition at line 1142 of file `specfun.h`.

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

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

See also

[legendre](#) for more details.

Definition at line 1152 of file `specfun.h`.

8.2.2.52 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::riemann_zeta (_Tp __s) [inline]`

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

The Riemann zeta function is defined by:

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

and

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

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 1223 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 1187 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 1197 of file `specfun.h`.

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

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

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 <code>n >= 0</code>
<code>__x</code>	The real argument <code>x >= 0</code>

Exceptions

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

Definition at line 1267 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 1238 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 1248 of file `specfun.h`.

8.2.2.58 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta) [inline]`

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

The spherical Legendre function is defined by

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

Template Parameters

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

Parameters

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

Definition at line 1314 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 1282 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 1293 of file `specfun.h`.

8.2.2.61 `template<typename __Tp> __gnu_cxx::__promote_fp_t<__Tp> std::sph_neumann (unsigned int __n, __Tp __x) [inline]`

Return the spherical Neumann function of integral order $n \geq 0$ 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 $x \geq 0$

Exceptions

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

Definition at line 1358 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 1329 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` $x \geq 0$.

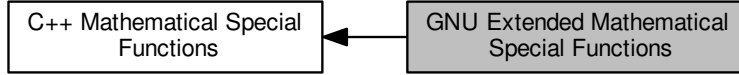
See also

[sph_neumann](#) for details.

Definition at line 1339 of file `specfun.h`.

8.3 GNU Extended Mathematical Special Functions

Collaboration diagram for GNU Extended Mathematical Special Functions:



Functions

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

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

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

The binomial coefficients are generated by:

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

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

- float [__gnu_cxx::binomialf](#) (unsigned int __n, unsigned int __k)
- long double [__gnu_cxx::binomiall](#) (unsigned int __n, unsigned int __k)
- template<typename _Tps, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tps, _Tp>](#) [__gnu_cxx::bose_einstein](#) (_Tps __s, _Tp __x)
- float [__gnu_cxx::bose_einsteinf](#) (float __s, float __x)
- long double [__gnu_cxx::bose_einsteinl](#) (long double __s, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::chebyshev_t](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_tf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_tl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::chebyshev_u](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_uf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_ul](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::chebyshev_v](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_vf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_vl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::chebyshev_w](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_wf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_wl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::clausen](#) (unsigned int __m, _Tp __w)
- template<typename _Tp >
std::complex< [__gnu_cxx::__promote_fp_t<_Tp>](#) > [__gnu_cxx::clausen](#) (unsigned int __m, std::complex< [__gnu_cxx::__promote_fp_t<_Tp>](#) > __z)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::clausen_cl](#) (unsigned int __m, _Tp __x)
- float [__gnu_cxx::clausen_clf](#) (unsigned int __m, float __x)
- long double [__gnu_cxx::clausen_cll](#) (unsigned int __m, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::clausen_sl](#) (unsigned int __m, _Tp __x)
- float [__gnu_cxx::clausen_slf](#) (unsigned int __m, float __x)
- long double [__gnu_cxx::clausen_sll](#) (unsigned int __m, long double __x)
- float [__gnu_cxx::clausenf](#) (unsigned int __m, float __x)
- std::complex< float > [__gnu_cxx::clausenf](#) (unsigned int __m, std::complex< float > __z)
- long double [__gnu_cxx::clausenl](#) (unsigned int __m, long double __x)
- std::complex< long double > [__gnu_cxx::clausenl](#) (unsigned int __m, std::complex< long double > __z)
- template<typename _Tk >
[__gnu_cxx::__promote_fp_t<_Tk>](#) [__gnu_cxx::comp_ellint_d](#) (_Tk __k)
- float [__gnu_cxx::comp_ellint_df](#) (float __k)
- long double [__gnu_cxx::comp_ellint_dl](#) (long double __k)
- float [__gnu_cxx::comp_ellint_rf](#) (float __x, float __y)
- long double [__gnu_cxx::comp_ellint_rl](#) (long double __x, long double __y)
- template<typename _Tx, typename _Ty >
[__gnu_cxx::__promote_fp_t<_Tx, _Ty>](#) [__gnu_cxx::comp_ellint_rf](#) (_Tx __x, _Ty __y)
- float [__gnu_cxx::comp_ellint_rg](#) (float __x, float __y)
- long double [__gnu_cxx::comp_ellint_rl](#) (long double __x, long double __y)
- template<typename _Tx, typename _Ty >
[__gnu_cxx::__promote_fp_t<_Tx, _Ty>](#) [__gnu_cxx::comp_ellint_rg](#) (_Tx __x, _Ty __y)

- `template<typename _Tpa, typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > __gnu_cxx::conf_hyperg (_Tpa __a, _Tpc __c, _Tp __x)`
- `template<typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpc, _Tp >::type __gnu_cxx::conf_hyperg_lim (_Tpc __c, _Tp __x)`
- `float __gnu_cxx::conf_hyperg_limf (float __c, float __x)`
- `long double __gnu_cxx::conf_hyperg_liml (long double __c, long double __x)`
- `float __gnu_cxx::conf_hypergf (float __a, float __c, float __x)`
- `long double __gnu_cxx::conf_hypergl (long double __a, long double __c, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::cos_pi (_Tp __x)`
- `float __gnu_cxx::cos_pif (float __x)`
- `long double __gnu_cxx::cos_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::cosh_pi (_Tp __x)`
- `float __gnu_cxx::cosh_pif (float __x)`
- `long double __gnu_cxx::cosh_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::coshint (_Tp __x)`
- `float __gnu_cxx::coshintf (float __x)`
- `long double __gnu_cxx::coshintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::cosint (_Tp __x)`
- `float __gnu_cxx::cosintf (float __x)`
- `long double __gnu_cxx::cosintl (long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::dawson (_Tp __x)`
- `float __gnu_cxx::dawsonf (float __x)`
- `long double __gnu_cxx::dawsonl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::debye (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::debyef (unsigned int __n, float __x)`
- `long double __gnu_cxx::debyel (unsigned int __n, long double __x)`

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::dilog (_Tp __x)`
- `float __gnu_cxx::dilogf (float __x)`
- `long double __gnu_cxx::dilogl (long double __x)`
- `template<typename _Tp >`
`_Tp __gnu_cxx::dirichlet_beta (_Tp __s)`
- `float __gnu_cxx::dirichlet_betaf (float __s)`
- `long double __gnu_cxx::dirichlet_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 >`
`_Tp __gnu_cxx::dirichlet_lambda (_Tp __s)`
- `float __gnu_cxx::dirichlet_lambdaf (float __s)`
- `long double __gnu_cxx::dirichlet_lambdaall (long double __s)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::double_factorial (int __n)`
Return the double factorial $n!!$ of the argument as a real number.

$$n!! = n(n-2)\dots(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)\dots(1), (-1)!! = 1$$

for odd n .
- `float __gnu_cxx::double_factorialf (int __n)`
- `long double __gnu_cxx::double_factoriall (int __n)`
- `template<typename _Tk, typename _Tp, typename _Ta, typename _Tb >`
`__gnu_cxx::__promote_fp_t< _Tk, _Tp, _Ta, _Tb > __gnu_cxx::ellint_cel (_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)`
- `float __gnu_cxx::ellint_celf (float __k_c, float __p, float __a, float __b)`
- `long double __gnu_cxx::ellint_cell (long double __k_c, long double __p, long double __a, long double __b)`
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::__promote_fp_t< _Tk, _Tphi > __gnu_cxx::ellint_d (_Tk __k, _Tphi __phi)`
- `float __gnu_cxx::ellint_df (float __k, float __phi)`
- `long double __gnu_cxx::ellint_dall (long double __k, long double __phi)`
- `template<typename _Tp, typename _Tk >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tk > __gnu_cxx::ellint_el1 (_Tp __x, _Tk __k_c)`
- `float __gnu_cxx::ellint_el1f (float __x, float __k_c)`
- `long double __gnu_cxx::ellint_el1all (long double __x, long double __k_c)`
- `template<typename _Tp, typename _Tk, typename _Ta, typename _Tb >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tk, _Ta, _Tb > __gnu_cxx::ellint_el2 (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)`
- `float __gnu_cxx::ellint_el2f (float __x, float __k_c, float __a, float __b)`
- `long double __gnu_cxx::ellint_el2all (long double __x, long double __k_c, long double __a, long double __b)`
- `template<typename _Tx, typename _Tk, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tx, _Tk, _Tp > __gnu_cxx::ellint_el3 (_Tx __x, _Tk __k_c, _Tp __p)`
- `float __gnu_cxx::ellint_el3f (float __x, float __k_c, float __p)`
- `long double __gnu_cxx::ellint_el3all (long double __x, long double __k_c, long double __p)`
- `template<typename _Tp, typename _Up >`
`__gnu_cxx::__promote_fp_t< _Tp, _Up > __gnu_cxx::ellint_rc (_Tp __x, _Up __y)`
- `float __gnu_cxx::ellint_rcf (float __x, float __y)`

- long double [__gnu_cxx::ellint_rcl](#) (long double __x, long double __y)
- template<typename _Tp, typename _Up, typename _Vp >
[__gnu_cxx::__promote_fp_t<_Tp, _Up, _Vp > __gnu_cxx::ellint_rd](#) (_Tp __x, _Up __y, _Vp __z)
- float [__gnu_cxx::ellint_rdf](#) (float __x, float __y, float __z)
- long double [__gnu_cxx::ellint_rdl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
[__gnu_cxx::__promote_fp_t<_Tp, _Up, _Vp > __gnu_cxx::ellint_rf](#) (_Tp __x, _Up __y, _Vp __z)
- float [__gnu_cxx::ellint_rff](#) (float __x, float __y, float __z)
- long double [__gnu_cxx::ellint_rfl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
[__gnu_cxx::__promote_fp_t<_Tp, _Up, _Vp > __gnu_cxx::ellint_rg](#) (_Tp __x, _Up __y, _Vp __z)
- float [__gnu_cxx::ellint_rgf](#) (float __x, float __y, float __z)
- long double [__gnu_cxx::ellint_rgl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp, typename _Wp >
[__gnu_cxx::__promote_fp_t<_Tp, _Up, _Vp, _Wp > __gnu_cxx::ellint_rj](#) (_Tp __x, _Up __y, _Vp __z, _Wp __p)
- float [__gnu_cxx::ellint_rjf](#) (float __x, float __y, float __z, float __p)
- long double [__gnu_cxx::ellint_rjl](#) (long double __x, long double __y, long double __z, long double __p)
- template<typename _Tp >
[_Tp __gnu_cxx::ellnome](#) (_Tp __k)
- float [__gnu_cxx::ellnomef](#) (float __k)
- long double [__gnu_cxx::ellnomel](#) (long double __k)
- template<typename _Tp >
[_Tp __gnu_cxx::euler](#) (unsigned int __n)

This returns Euler number E_n .

- template<typename _Tp >
[_Tp __gnu_cxx::eulerian_1](#) (unsigned int __n, unsigned int __m)
- template<typename _Tp >
[_Tp __gnu_cxx::eulerian_2](#) (unsigned int __n, unsigned int __m)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp > __gnu_cxx::expint](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::expintf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::expintl](#) (unsigned int __n, long double __x)
- template<typename _Tlam, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tlam, _Tp > __gnu_cxx::exponential_cdf](#) (_Tlam __lambda, _Tp __x)
- template<typename _Tlam, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tlam, _Tp > __gnu_cxx::exponential_pdf](#) (_Tlam __lambda, _Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp > __gnu_cxx::factorial](#) (unsigned int __n)

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

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

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

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

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

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

- float [__gnu_cxx::falling_factorialf](#) (float __a, float __nu)
- long double [__gnu_cxx::falling_factoriall](#) (long double __a, long double __nu)
- template<typename _Tps, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tps, _Tp>](#) [__gnu_cxx::fermi_dirac](#) (_Tps __s, _Tp __x)
- float [__gnu_cxx::fermi_diracf](#) (float __s, float __x)
- long double [__gnu_cxx::fermi_diracl](#) (long double __s, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::fisher_f_cdf](#) (_Tp __F, unsigned int __nu1, unsigned int __nu2)
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::fisher_f_pdf](#) (_Tp __F, unsigned int __nu1, unsigned int __nu2)
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::fresnel_c](#) (_Tp __x)
- float [__gnu_cxx::fresnel_cf](#) (float __x)
- long double [__gnu_cxx::fresnel_cl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::fresnel_s](#) (_Tp __x)
- float [__gnu_cxx::fresnel_sf](#) (float __x)
- long double [__gnu_cxx::fresnel_sl](#) (long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp>](#) [__gnu_cxx::gamma_cdf](#) (_Ta __alpha, _Tb __beta, _Tp __x)
Return the gamma cumulative propability distribution function.
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp>](#) [__gnu_cxx::gamma_pdf](#) (_Ta __alpha, _Tb __beta, _Tp __x)
Return the gamma propability distribution function.
- template<typename _Ta >
[__gnu_cxx::__promote_fp_t<_Ta>](#) [__gnu_cxx::gamma_reciprocal](#) (_Ta __a)
- float [__gnu_cxx::gamma_reciprocalf](#) (float __a)
- long double [__gnu_cxx::gamma_reciprocall](#) (long double __a)
- template<typename _Talpha, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Talpha, _Tp>](#) [__gnu_cxx::gegenbauer](#) (unsigned int __n, _Talpha __alpha, _Tp __x)
- float [__gnu_cxx::gegenbauerf](#) (unsigned int __n, float __alpha, float __x)
- long double [__gnu_cxx::gegenbauerl](#) (unsigned int __n, long double __alpha, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::harmonic](#) (unsigned int __n)
- template<typename _Tk, typename _Tphi >
[__gnu_cxx::__promote_fp_t<_Tk, _Tphi>](#) [__gnu_cxx::heuman_lambda](#) (_Tk __k, _Tphi __phi)
- float [__gnu_cxx::heuman_lambdaf](#) (float __k, float __phi)
- long double [__gnu_cxx::heuman_lambdal](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Up >
[__gnu_cxx::__promote_fp_t<_Tp, _Up>](#) [__gnu_cxx::hurwitz_zeta](#) (_Tp __s, _Up __a)

- `template<typename _Tp, typename _Up >`
`std::complex< _Tp > __gnu_cxx::hurwitz_zeta (_Tp __s, std::complex< _Up > __a)`
- `float __gnu_cxx::hurwitz_zetaf (float __s, float __a)`
- `long double __gnu_cxx::hurwitz_zetal (long double __s, long double __a)`
- `template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpa, _Tpb, _Tpc, _Tp > __gnu_cxx::hyperg (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)`
- `float __gnu_cxx::hypergfi (float __a, float __b, float __c, float __x)`
- `long double __gnu_cxx::hypergli (long double __a, long double __b, long double __c, long double __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta (_Ta __a, _Tb __b, _Tp __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::ibetaci (_Ta __a, _Tb __b, _Tp __x)`
- `float __gnu_cxx::ibetacfi (float __a, float __b, float __x)`
- `long double __gnu_cxx::ibetacil (long double __a, long double __b, long double __x)`
- `float __gnu_cxx::ibetafi (float __a, float __b, float __x)`
- `long double __gnu_cxx::ibetal (long double __a, long double __b, long double __x)`
- `template<typename _Talpha, typename _Tbeta, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_fp_t< _Kp, _Up > __gnu_cxx::jacobi_cn (_Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_cnf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_cnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_fp_t< _Kp, _Up > __gnu_cxx::jacobi_dn (_Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_dnf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_dnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_fp_t< _Kp, _Up > __gnu_cxx::jacobi_sn (_Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_snfi (float __k, float __u)`
- `long double __gnu_cxx::jacobi_snl (long double __k, long double __u)`
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::__promote_fp_t< _Tk, _Tphi > __gnu_cxx::jacobi_zeta (_Tk __k, _Tphi __phi)`
- `float __gnu_cxx::jacobi_zetafi (float __k, float __phi)`
- `long double __gnu_cxx::jacobi_zetal (long double __k, long double __phi)`
- `float __gnu_cxx::jacobi (unsigned __n, float __alpha, float __beta, float __x)`
- `long double __gnu_cxx::jacobil (unsigned __n, long double __alpha, long double __beta, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::lbinomial (unsigned int __n, unsigned int __k)`

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

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

The binomial coefficients are generated by:

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

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

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

$$n!! = n(n-2)\dots(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)\dots(1), (-1)!! = 1$$

for odd n .

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

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

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

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

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

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

In particular, $n! = n! \cdot 1$. Thus this function returns

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

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

$$\left\{ \begin{matrix} a \\ \nu \end{matrix} \right\}$$

, and others.

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

Return the logistic cumulative distribution function.

- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::logistic_pdf (_Ta __a, _Tb __b, _Tp __x)`
Return the logistic probability density function.
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::lognormal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the lognormal cumulative probability density function.
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::lognormal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the lognormal probability density function.
- `template<typename _Tp, typename _Tnu >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::lrising_factorial (_Tp __a, _Tnu __nu)`
Return the logarithm of the rising factorial function or the (upper) Pochhammer symbol. The rising factorial function is defined for integer order by

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

Thus this function returns

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

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

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

, and others.

- `float __gnu_cxx::lrising_factorialf (float __a, float __nu)`
- `long double __gnu_cxx::lrising_factoriall (long double __a, long double __nu)`
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::normal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the normal cumulative probability density function.
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::normal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the normal probability density function.
- `template<typename _Tph, typename _Tpa >`
`__gnu_cxx::__promote_fp_t< _Tph, _Tpa > __gnu_cxx::owens_t (_Tph __h, _Tpa __a)`
- `float __gnu_cxx::owens_tf (float __h, float __a)`
- `long double __gnu_cxx::owens_tl (long double __h, long double __a)`
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::pgamma (_Ta __a, _Tp __x)`
- `float __gnu_cxx::pgammaf (float __a, float __x)`
- `long double __gnu_cxx::pgammal (long double __a, long double __x)`
- `template<typename _Tp, typename _Wp >`
`__gnu_cxx::__promote_fp_t< _Tp, _Wp > __gnu_cxx::polylog (_Tp __s, _Wp __w)`
- `template<typename _Tp, typename _Wp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tp, _Wp > > __gnu_cxx::polylog (_Tp __s, std::complex< _Tp > __w)`
- `float __gnu_cxx::polylogf (float __s, float __w)`
- `std::complex< float > __gnu_cxx::polylogf (float __s, std::complex< float > __w)`
- `long double __gnu_cxx::polylogl (long double __s, long double __w)`
- `std::complex< long double > __gnu_cxx::polylogl (long double __s, std::complex< long double > __w)`

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::psi (_Tp __x)`
- `float __gnu_cxx::psif (float __x)`
- `long double __gnu_cxx::psil (long double __x)`
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::qgamma (_Ta __a, _Tp __x)`
- `float __gnu_cxx::qgammaf (float __a, float __x)`
- `long double __gnu_cxx::qgamma (long double __a, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::radpoly (unsigned int __n, unsigned int __m, _Tp __rho)`
- `float __gnu_cxx::radpolyf (unsigned int __n, unsigned int __m, float __rho)`
- `long double __gnu_cxx::radpolyl (unsigned int __n, unsigned int __m, long double __rho)`
- `template<typename _Tp, typename _Tnu >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::rising_factorial (_Tp __a, _Tnu __nu)`

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

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

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

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

, and others.

- `float __gnu_cxx::rising_factorialf (float __a, float __nu)`
- `long double __gnu_cxx::rising_factoriall (long double __a, long double __nu)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sin_pi (_Tp __x)`
- `float __gnu_cxx::sin_pif (float __x)`
- `long double __gnu_cxx::sin_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinc (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinc_pi (_Tp __x)`
- `float __gnu_cxx::sinc_pif (float __x)`
- `long double __gnu_cxx::sinc_pil (long double __x)`
- `float __gnu_cxx::sincf (float __x)`
- `long double __gnu_cxx::sincl (long double __x)`
- `__gnu_cxx::__sincos_t< double > __gnu_cxx::sincos (double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sincos (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sincos_pi (_Tp __x)`
- `__gnu_cxx::__sincos_t< float > __gnu_cxx::sincos_pif (float __x)`
- `__gnu_cxx::__sincos_t< long double > __gnu_cxx::sincos_pil (long double __x)`
- `__gnu_cxx::__sincos_t< float > __gnu_cxx::sincosf (float __x)`
- `__gnu_cxx::__sincos_t< long double > __gnu_cxx::sincosl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinh_pi (_Tp __x)`
- `float __gnu_cxx::sinh_pif (float __x)`
- `long double __gnu_cxx::sinh_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinhc (_Tp __x)`

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinhc_pi (_Tp __x)`
- `float __gnu_cxx::sinhc_pif (float __x)`
- `long double __gnu_cxx::sinhc_pil (long double __x)`
- `float __gnu_cxx::sinhcf (float __x)`
- `long double __gnu_cxx::sinhcl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinhint (_Tp __x)`
- `float __gnu_cxx::sinhintf (float __x)`
- `long double __gnu_cxx::sinhintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sinint (_Tp __x)`
- `float __gnu_cxx::sinintf (float __x)`
- `long double __gnu_cxx::sinintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sph_bessel_i (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::sph_bessel_if (unsigned int __n, float __x)`
- `long double __gnu_cxx::sph_bessel_il (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::sph_bessel_k (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::sph_bessel_kf (unsigned int __n, float __x)`
- `long double __gnu_cxx::sph_bessel_kl (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, _Tp __z)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sph_hankel_1 (unsigned int __n, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, float __z)`
- `std::complex< float > __gnu_cxx::sph_hankel_1f (unsigned int __n, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, long double __z)`
- `std::complex< long double > __gnu_cxx::sph_hankel_1l (unsigned int __n, std::complex< long double > __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, _Tp __z)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sph_hankel_2 (unsigned int __n, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, float __z)`
- `std::complex< float > __gnu_cxx::sph_hankel_2f (unsigned int __n, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, long double __z)`
- `std::complex< long double > __gnu_cxx::sph_hankel_2l (unsigned int __n, std::complex< long double > __x)`
- `template<typename _Ttheta, typename _Tphi >`
`std::complex< __gnu_cxx::__promote_fp_t< _Ttheta, _Tphi > > __gnu_cxx::sph_harmonic (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)`
- `std::complex< float > __gnu_cxx::sph_harmonicf (unsigned int __l, int __m, float __theta, float __phi)`
- `std::complex< long double > __gnu_cxx::sph_harmonicl (unsigned int __l, int __m, long double __theta, long double __phi)`
- `template<typename _Tp >`
`_Tp __gnu_cxx::stirling_1 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __gnu_cxx::stirling_2 (unsigned int __n, unsigned int __m)`
- `template<typename _Tt, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::student_t_cdf (_Tt __t, unsigned int __nu)`

Return the Students T probability function.

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

Return the complement of the Students T probability function.

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::tan_pi (_Tp __x)`
- `float __gnu_cxx::tan_pif (float __x)`
- `long double __gnu_cxx::tan_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::tanh_pi (_Tp __x)`
- `float __gnu_cxx::tanh_pif (float __x)`
- `long double __gnu_cxx::tanh_pil (long double __x)`
- `template<typename _Ta >`
`__gnu_cxx::__promote_fp_t< _Ta > __gnu_cxx::tgamma (_Ta __a)`
- `template<typename _Ta >`
`std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::tgamma (std::complex< _Ta > __a)`
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma (_Ta __a, _Tp __x)`
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma_lower (_Ta __a, _Tp __x)`
- `float __gnu_cxx::tgamma_lowerf (float __a, float __x)`
- `long double __gnu_cxx::tgamma_lowerl (long double __a, long double __x)`
- `float __gnu_cxx::tgammaf (float __a)`
- `std::complex< float > __gnu_cxx::tgammaf (std::complex< float > __a)`
- `float __gnu_cxx::tgammaf (float __a, float __x)`
- `long double __gnu_cxx::tgammal (long double __a)`
- `std::complex< long double > __gnu_cxx::tgammal (std::complex< long double > __a)`
- `long double __gnu_cxx::tgammal (long double __a, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_1 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_1f (float __nu, float __x)`
- `long double __gnu_cxx::theta_1l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_2 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_2f (float __nu, float __x)`
- `long double __gnu_cxx::theta_2l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_3 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_3f (float __nu, float __x)`
- `long double __gnu_cxx::theta_3l (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_4 (_Tpnu __nu, _Tp __x)`
- `float __gnu_cxx::theta_4f (float __nu, float __x)`
- `long double __gnu_cxx::theta_4l (long double __nu, long double __x)`
- `template<typename _Tpk, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpk, _Tp > __gnu_cxx::theta_c (_Tpk __k, _Tp __x)`
- `float __gnu_cxx::theta_cf (float __k, float __x)`
- `long double __gnu_cxx::theta_cl (long double __k, long double __x)`
- `template<typename _Tpk, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpk, _Tp > __gnu_cxx::theta_d (_Tpk __k, _Tp __x)`
- `float __gnu_cxx::theta_df (float __k, float __x)`

- long double `__gnu_cxx::theta_dl` (long double `__k`, long double `__x`)
- `template<typename _Tpk, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpk, _Tp > __gnu_cxx::theta_n` (`_Tpk __k`, `_Tp __x`)
- float `__gnu_cxx::theta_nf` (float `__k`, float `__x`)
- long double `__gnu_cxx::theta_nl` (long double `__k`, long double `__x`)
- `template<typename _Tpk, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpk, _Tp > __gnu_cxx::theta_s` (`_Tpk __k`, `_Tp __x`)
- float `__gnu_cxx::theta_sf` (float `__k`, float `__x`)
- long double `__gnu_cxx::theta_sl` (long double `__k`, long double `__x`)
- `template<typename _Tpa, typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > __gnu_cxx::tricomi_u` (`_Tpa __a`, `_Tpc __c`, `_Tp __x`)
- float `__gnu_cxx::tricomi_uf` (float `__a`, float `__c`, float `__x`)
- long double `__gnu_cxx::tricomi_ul` (long double `__a`, long double `__c`, long double `__x`)
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::weibull_cdf` (`_Ta __a`, `_Tb __b`, `_Tp __x`)
Return the Weibull cumulative probability density function.
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::weibull_pdf` (`_Ta __a`, `_Tb __b`, `_Tp __x`)
Return the Weibull probability density function.
- `template<typename _Trho, typename _Tphi >`
`__gnu_cxx::__promote_fp_t< _Trho, _Tphi > __gnu_cxx::zernike` (unsigned int `__n`, int `__m`, `_Trho __rho`, `_Tphi __phi`)
- float `__gnu_cxx::zernikef` (unsigned int `__n`, int `__m`, float `__rho`, float `__phi`)
- long double `__gnu_cxx::zernikel` (unsigned int `__n`, int `__m`, long double `__rho`, long double `__phi`)

8.3.1 Detailed Description

An extended collection of advanced mathematical special functions for GNU.

8.3.2 Function Documentation

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

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

The Airy function is defined by:

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

Template Parameters

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

Parameters

\leftrightarrow	The argument
x	

Definition at line 2799 of file specfun.h.

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

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

The Airy function is defined by:

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

Template Parameters

Tp	The real type of the argument
------	-------------------------------

Parameters

\leftrightarrow	The complex argument
x	

Definition at line 2819 of file specfun.h.

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

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

See also

[airy_ai](#) for details.

Definition at line 2772 of file specfun.h.

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

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

See also

[airy_ai](#) for details.

Definition at line 2782 of file specfun.h.

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

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

The Airy function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 2861 of file `specfun.h`.

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

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

The Airy function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 2882 of file `specfun.h`.

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

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

See also

[airy_bi](#) for details.

Definition at line 2833 of file specfun.h.

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

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

See also

[airy_bi](#) for details.

Definition at line 2843 of file specfun.h.

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

Return the Bernoulli number of integer order n .

The Bernoulli numbers are defined by

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

All odd Bernoulli numbers except B_1 are zero.

Parameters

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

Definition at line 4254 of file specfun.h.

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

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

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

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

The derivative is proportional to the previous polynomial:

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

The series expansion for the Bernoulli polynomials is:

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

A useful argument promotion is:

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

Definition at line 6576 of file specfun.h.

References std::__detail::__bernoulli().

8.3.2.11 `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 4227 of file specfun.h.

8.3.2.12 `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 4237 of file specfun.h.

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

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

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

The binomial coefficients are generated by:

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

.

Parameters

\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 4170 of file specfun.h.

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

Return the binomial cumulative distribution function.

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

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

Parameters

\leftrightarrow _p	
\leftrightarrow _n	
\leftrightarrow _k	

Definition at line 6429 of file specfun.h.

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

Return the binomial probability mass function.

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

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

Parameters

\leftrightarrow _p	
\leftrightarrow _n	
\leftrightarrow _k	

Definition at line 6408 of file specfun.h.

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

Return the binomial coefficient as a `float`.

See also

[binomial](#) for details.

Definition at line 4141 of file specfun.h.

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

Return the binomial coefficient as a `long double`.

See also

[binomial](#) for details.

Definition at line 4150 of file specfun.h.

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

Definition at line 5807 of file specfun.h.

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

Definition at line 5798 of file specfun.h.

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

Definition at line 5802 of file specfun.h.

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

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

The Chebyshev polynomial of the first kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 2044 of file `specfun.h`.

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

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

See also

[chebyshev_t](#) for details.

Definition at line 2015 of file `specfun.h`.

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

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

See also

[chebyshev_t](#) for details.

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

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

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

The Chebyshev polynomial of the second kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 2088 of file `specfun.h`.

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

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

See also

[chebyshev_u](#) for details.

Definition at line 2059 of file `specfun.h`.

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

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

See also

[chebyshev_u](#) for details.

Definition at line 2069 of file `specfun.h`.

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

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

The Chebyshev polynomial of the third kind is defined by:

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

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

Template Parameters

<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 2133 of file `specfun.h`.

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

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

See also

[chebyshev_v](#) for details.

Definition at line 2103 of file `specfun.h`.

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

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

See also

[chebyshev_v](#) for details.

Definition at line 2113 of file `specfun.h`.

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

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

The Chebyshev polynomial of the fourth kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 2178 of file `specfun.h`.

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

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

See also

[chebyshev_w](#) for details.

Definition at line 2148 of file `specfun.h`.

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

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

See also

[chebyshev_w](#) for details.

Definition at line 2158 of file `specfun.h`.

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

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

The Clausen function is defined by

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

Template Parameters

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

Parameters

<code>↵ _m</code>	The integral order
<code>↵ _x</code>	The real argument

Definition at line 5282 of file specfun.h.

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

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

The Clausen function is defined by

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

Template Parameters

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

Parameters

<code>↵ _m</code>	The integral order
<code>↵ _z</code>	The complex argument

Definition at line 5326 of file specfun.h.

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

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

The Clausen cosine function is defined by

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

Template Parameters

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

Parameters

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

Definition at line 5238 of file specfun.h.

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

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

See also

[clausen_cl](#) for details.

Definition at line 5210 of file specfun.h.

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

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

See also

[clausen_cl](#) for details.

Definition at line 5220 of file specfun.h.

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

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

The Clausen sine function is defined by

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

Template Parameters

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

Parameters

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

Definition at line 5195 of file specfun.h.

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

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

See also

[clausen_sl](#) for details.

Definition at line 5167 of file specfun.h.

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

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

See also

[clausen_sl](#) for details.

Definition at line 5177 of file specfun.h.

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

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

See also

[clausen](#) for details.

Definition at line 5253 of file specfun.h.

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

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

See also

[clausen](#) for details.

Definition at line 5297 of file `specfun.h`.

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

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

See also

[clausen](#) for details.

Definition at line 5263 of file `specfun.h`.

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

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

See also

[clausen](#) for details.

Definition at line 5307 of file `specfun.h`.

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

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

The complete Legendre elliptic integral D is defined by

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

Template Parameters

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

Parameters

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

Definition at line 4456 of file specfun.h.

8.3.2.46 `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 4429 of file specfun.h.

8.3.2.47 `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 4439 of file specfun.h.

8.3.2.48 `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 3142 of file specfun.h.

8.3.2.49 `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 3152 of file specfun.h.

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

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

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

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

Parameters

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

Definition at line 3170 of file specfun.h.

8.3.2.51 `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 3375 of file specfun.h.

8.3.2.52 `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 3384 of file specfun.h.

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

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

The complete Carlson elliptic function is defined by:

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

Parameters

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

Definition at line 3403 of file specfun.h.

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

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

The confluent hypergeometric function is defined by

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

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

Parameters

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

Definition at line 1423 of file specfun.h.

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

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

The confluent hypergeometric limit function is defined by

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

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

Parameters

\leftrightarrow _c	The denominatorial parameter
\leftrightarrow _x	The argument

Definition at line 1568 of file specfun.h.

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

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

See also

[conf_hyperg_lim](#) for details.

Definition at line 1539 of file `specfun.h`.

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

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

See also

[conf_hyperg_lim](#) for details.

Definition at line 1549 of file `specfun.h`.

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

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

See also

[conf_hyperg](#) for details.

Definition at line 1391 of file `specfun.h`.

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

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

See also

[conf_hyperg](#) for details.

Definition at line 1402 of file `specfun.h`.

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

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

The reperiodized cosine function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 5933 of file `specfun.h`.

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

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

See also

[cos_pi](#) for more details.

Definition at line 5906 of file `specfun.h`.

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

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

See also

[cos_pi](#) for more details.

Definition at line 5916 of file `specfun.h`.

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

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

The reperiodized hyperbolic cosine function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 5975 of file specfun.h.

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

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

See also

[cosh_pi](#) for more details.

Definition at line 5948 of file specfun.h.

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

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

See also

[cosh_pi](#) for more details.

Definition at line 5958 of file specfun.h.

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

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

The hyperbolic cosine integral is defined by

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

Template Parameters

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

Parameters

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

Definition at line 1850 of file specfun.h.

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

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

See also

[coshint](#) for details.

Definition at line 1822 of file specfun.h.

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

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

See also

[coshint](#) for details.

Definition at line 1832 of file specfun.h.

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

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

The cosine integral is defined by

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

Parameters

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

Definition at line 1767 of file specfun.h.

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

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

See also

[cosint](#) for details.

Definition at line 1741 of file `specfun.h`.

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

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

See also

[cosint](#) for details.

Definition at line 1751 of file `specfun.h`.

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

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

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

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

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

See also

`cyl_bessel` and `cyl_neumann`).

Template Parameters

<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 2526 of file specfun.h.

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

Return the complex cylindrical Hankel function of the first kind $H_\nu^{(1)}(x)$ of complex order 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 4733 of file specfun.h.

8.3.2.74 `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 2494 of file specfun.h.

8.3.2.75 `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 4702 of file specfun.h.

8.3.2.76 `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 2505 of file specfun.h.

8.3.2.77 `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 4713 of file specfun.h.

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

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

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

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

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

See also

[cyl_bessel](#) and [cyl_neumann](#)).

Template Parameters

<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 2574 of file specfun.h.

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

Return the complex cylindrical Hankel function of the second kind $H_\nu^{(2)}(x)$ of complex order 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 4780 of file specfun.h.

```
8.3.2.80  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 2542 of file specfun.h.

```
8.3.2.81  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 4749 of file specfun.h.

8.3.2.82 `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 2553 of file `specfun.h`.

8.3.2.83 `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 4760 of file `specfun.h`.

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

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

The Dawson integral is defined by:

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

and it's derivative is:

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

Parameters

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

Definition at line 3744 of file `specfun.h`.

8.3.2.85 `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 3715 of file specfun.h.

8.3.2.86 `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 3725 of file specfun.h.

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

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

The Debye function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 6545 of file specfun.h.

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

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

See also

[debye](#) for details.

Definition at line 6517 of file specfun.h.

8.3.2.89 `long double __gnu_cxx::debye1 (unsigned int __n, long double __x) [inline]`

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

See also

[debye](#) for details.

Definition at line 6527 of file specfun.h.

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

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

The dilogarithm is defined by:

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

Parameters

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

Definition at line 3127 of file specfun.h.

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

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

See also

[dilog](#) for details.

Definition at line 3101 of file specfun.h.

8.3.2.92 `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 3111 of file specfun.h.

8.3.2.93 `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:

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

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

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

Parameters

\longleftrightarrow	
<code>__s</code>	

Definition at line 5109 of file specfun.h.

8.3.2.94 `float __gnu_cxx::dirichlet_betaf (float __s) [inline]`

Return the Dirichlet beta function of real argument s .

See also

[dirichlet_beta](#) for details.

Definition at line 5074 of file specfun.h.

8.3.2.95 `long double __gnu_cxx::dirichlet_betalf (long double __s) [inline]`

Return the Dirichlet beta function of real argument s .

See also

[dirichlet_beta](#) for details.

Definition at line 5083 of file specfun.h.

8.3.2.96 `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:

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

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

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

Parameters

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

Definition at line 5060 of file specfun.h.

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

Return the Dirichlet eta function of real argument s .

See also

[dirichlet_eta](#) for details.

Definition at line 5024 of file specfun.h.

8.3.2.98 `long double __gnu_cxx::dirichlet_etald (long double __s) [inline]`

Return the Dirichlet eta function of real argument s .

See also

[dirichlet_eta](#) for details.

Definition at line 5033 of file specfun.h.

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

Return the Dirichlet lambda function of real argument s .

The Dirichlet lambda function is defined by

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

In terms of the Riemann zeta and the Dirichlet eta functions

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

Parameters

\leftarrow	
<code>__s</code>	

Definition at line 5152 of file specfun.h.

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

Return the Dirichlet lambda function of real argument s .

See also

[dirichlet_lambda](#) for details.

Definition at line 5123 of file specfun.h.

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

Return the Dirichlet lambda function of real argument s .

See also

[dirichlet_lambda](#) for details.

Definition at line 5132 of file specfun.h.

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

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

$$n!! = n(n-2)\dots(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)\dots(1), (-1)!! = 1$$

for odd n .

Definition at line 4048 of file `specfun.h`.

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

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

See also

[double_factorial](#) for more details

Definition at line 4021 of file `specfun.h`.

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

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

See also

[double_factorial](#) for more details

Definition at line 4031 of file `specfun.h`.

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

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

The Bulirsch complete elliptic integral is defined by

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

Parameters

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

Definition at line 4686 of file specfun.h.

8.3.2.106 `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 4654 of file specfun.h.

8.3.2.107 `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 4663 of file specfun.h.

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

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

The Legendre elliptic integral D is defined by

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

Parameters

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

Definition at line 4499 of file `specfun.h`.

8.3.2.109 `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 4471 of file `specfun.h`.

8.3.2.110 `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 4481 of file `specfun.h`.

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

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

The Bulirsch elliptic integral of the first kind is defined by

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

Parameters

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

Definition at line 4545 of file specfun.h.

8.3.2.112 `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 4515 of file specfun.h.

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

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

See also

[ellint_el1](#) for details.

Definition at line 4526 of file specfun.h.

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

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

The Bulirsch elliptic integral of the second kind is defined by

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

Parameters

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

Definition at line 4591 of file specfun.h.

8.3.2.115 `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 4560 of file specfun.h.

8.3.2.116 `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 4570 of file specfun.h.

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

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

The Bulirsch elliptic integral of the third kind is defined by

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

Parameters

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

Definition at line 4638 of file specfun.h.

8.3.2.118 `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 4607 of file `specfun.h`.

8.3.2.119 `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 4618 of file `specfun.h`.

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

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

The Carlson elliptic function is defined by:

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 3262 of file specfun.h.

8.3.2.121 `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 3228 of file specfun.h.

8.3.2.122 `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 3237 of file specfun.h.

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

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

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

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 3361 of file specfun.h.

8.3.2.124 `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 3325 of file specfun.h.

8.3.2.125 `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 3334 of file specfun.h.

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

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

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

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

Parameters

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

Definition at line 3214 of file specfun.h.

8.3.2.127 `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 3185 of file `specfun.h`.

8.3.2.128 `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 3195 of file `specfun.h`.

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

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

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

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 3452 of file specfun.h.

8.3.2.130 `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 3417 of file specfun.h.

8.3.2.131 `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 3426 of file specfun.h.

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

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

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

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 3311 of file specfun.h.

8.3.2.133 `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 3276 of file specfun.h.

8.3.2.134 `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 3285 of file specfun.h.

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

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

The elliptic nome function is defined by

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

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

Template Parameters

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

Parameters

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

Definition at line 5540 of file specfun.h.

8.3.2.136 `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 5513 of file specfun.h.

8.3.2.137 `long double __gnu_cxx::ellnome1 (long double __k) [inline]`

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

See also

[ellnome](#) for details.

Definition at line 5523 of file specfun.h.

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

This returns Euler number E_n .

Parameters

<code>__n</code>	the order n of the Euler number.
------------------	----------------------------------

Returns

The Euler number of order n.

Definition at line 6587 of file specfun.h.

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

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

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

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

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

Definition at line 6605 of file specfun.h.

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

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

$$\left\langle\left\langle n \right\rangle\right\rangle_m = (2n - m - 1) \left\langle\left\langle n - 1 \right\rangle\right\rangle_{m-1} + (m + 1) \left\langle\left\langle n - 1 \right\rangle\right\rangle_m \text{ for } n > 0$$

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

Definition at line 6623 of file specfun.h.

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

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

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

In particular

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

Template Parameters

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

Parameters

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

Definition at line 3790 of file specfun.h.

8.3.2.142 `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 3759 of file specfun.h.

8.3.2.143 `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 3769 of file `specfun.h`.

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

Return the exponential cumulative probability density function.

The formula for the exponential cumulative probability density function is

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

Definition at line 6264 of file `specfun.h`.

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

Return the exponential probability density function.

The formula for the exponential probability density function is

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

Definition at line 6248 of file `specfun.h`.

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

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

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

.

Definition at line 4007 of file `specfun.h`.

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

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

See also

[factorial](#) for more details

Definition at line 3987 of file `specfun.h`.

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

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

See also

[factorial](#) for more details

Definition at line 3996 of file `specfun.h`.

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

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

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

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

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

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

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

See also

[falling_factorial](#) for details.

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

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

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

See also

[falling_factorial](#) for details.

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

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

Definition at line 5789 of file `specfun.h`.

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

Definition at line 5780 of file `specfun.h`.

8.3.2.154 `long double __gnu_cxx::fermi_dirac1(long double __s, long double __x)` `[inline]`

Definition at line 5784 of file `specfun.h`.

8.3.2.155 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::fisher_f_cdf(_Tp __F, unsigned int __nu1, unsigned int __nu2)`

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .

The f-distribution propability function is related to the incomplete beta function:

$$Q(F|\nu_1, \nu_2) = I_{\frac{\nu_2}{\nu_2 + \nu_1 F}}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right)$$

Parameters

<code>__nu1</code>	The number of degrees of freedom of sample 1
<code>__nu2</code>	The number of degrees of freedom of sample 2
<code>__F</code>	The F statistic

Definition at line 6362 of file `specfun.h`.

8.3.2.156 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::fisher_f_pdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .

The f-distribution propability function is related to the incomplete beta function:

$$P(F|\nu_1, \nu_2) = 1 - I_{\frac{\nu_2}{\nu_2 + \nu_1 F}}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right) = 1 - Q(F|\nu_1, \nu_2)$$

Parameters

<code>__F</code>	
<code>__nu1</code>	
<code>__nu2</code>	

Definition at line 6387 of file specfun.h.

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

Return the Fresnel cosine integral of argument x .

The Fresnel cosine integral is defined by

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

Parameters

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

Definition at line 3701 of file specfun.h.

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

Definition at line 3682 of file specfun.h.

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

Definition at line 3686 of file specfun.h.

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

Return the Fresnel sine integral of argument x .

The Fresnel sine integral is defined by

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

Parameters

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

Definition at line 3673 of file `specfun.h`.

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

Definition at line 3654 of file `specfun.h`.

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

Definition at line 3658 of file `specfun.h`.

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

Return the gamma cumulative propability distribution function.

The formula for the gamma probability density function is:

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

Definition at line 6166 of file `specfun.h`.

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

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

Return the gamma propability distribution function.

The formula for the gamma probability density function is:

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

Definition at line 6149 of file `specfun.h`.

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

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

Return the reciprocal gamma function for real argument.

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

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

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

Definition at line 6502 of file `specfun.h`.

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

Return the reciprocal gamma function for `float` argument.

See also

[gamma_reciprocal](#) for details.

Definition at line 6477 of file `specfun.h`.

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

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

See also

[gamma_reciprocal](#) for details.

Definition at line 6487 of file `specfun.h`.

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

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

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

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

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

Template Parameters

<code>__Talpha</code>	The real type of the order
<code>__Tp</code>	The real type of the argument

Parameters

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

Definition at line 2286 of file specfun.h.

8.3.2.169 `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 2253 of file specfun.h.

8.3.2.170 `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 2264 of file specfun.h.

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

Return the harmonic number H_n .

The the harmonic number is defined by

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

Parameters

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

Definition at line 3565 of file specfun.h.

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

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

The complete Heuman lambda function is defined by

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

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

Template Parameters

<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 4414 of file specfun.h.

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

Definition at line 4388 of file specfun.h.

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

Definition at line 4392 of file specfun.h.

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

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

The the Hurwitz zeta function is defined by

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

Parameters

\leftrightarrow _s	The argument
\leftrightarrow _a	The parameter

Definition at line 3493 of file specfun.h.

```
8.3.2.176 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 3507 of file specfun.h.

```
8.3.2.177 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 3467 of file specfun.h.

```
8.3.2.178 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 3477 of file specfun.h.

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

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

The hypergeometric function is defined by

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

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

Parameters

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

Definition at line 1522 of file specfun.h.

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

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

See also

[hyperg](#) for details.

Definition at line 1489 of file specfun.h.

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

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

See also

[hyperg](#) for details.

Definition at line 1500 of file specfun.h.

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

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

The regularized incomplete beta function is defined by

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

where

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

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

Parameters

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

Definition at line 3614 of file specfun.h.

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

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

The regularized complementary incomplete beta function is defined by

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

Parameters

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

Definition at line 3645 of file specfun.h.

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

Definition at line 3623 of file specfun.h.

References `__gnu_cxx::ibetaf()`.

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

Definition at line 3627 of file specfun.h.

References `__gnu_cxx::ibetal()`.

8.3.2.186 `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 3580 of file `specfun.h`.

Referenced by `__gnu_cxx::ibetaf()`.

8.3.2.187 `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 3590 of file `specfun.h`.

Referenced by `__gnu_cxx::ibetac()`.

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

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

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

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

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

Template Parameters

<code>_Talpha</code>	The real type of the order 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 2238 of file `specfun.h`.

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

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

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

The Jacobi elliptic `cn` integral is defined by

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

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

See also

`ellint_1`).

Template Parameters

<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 1950 of file `specfun.h`.

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

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

See also

[jacobi_cn](#) for details.

Definition at line 1915 of file `specfun.h`.

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

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

See also

[jacobi_cn](#) for details.

Definition at line 1927 of file `specfun.h`.

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

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

The Jacobi elliptic `dn` integral is defined by

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

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

See also

`ellint_1`).

Template Parameters

<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 2000 of file `specfun.h`.

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

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

See also

[jacobi_dn](#) for details.

Definition at line 1965 of file `specfun.h`.

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

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

See also

[jacobi_dn](#) for details.

Definition at line 1977 of file `specfun.h`.

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

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

The Jacobi elliptic `sn` integral is defined by

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

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

See also

`ellint_1`).

Template Parameters

<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 1900 of file `specfun.h`.

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

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

See also

[jacobi_sn](#) for details.

Definition at line 1865 of file `specfun.h`.

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

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

See also

[jacobi_sn](#) for details.

Definition at line 1877 of file `specfun.h`.

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

Return the Jacobi zeta function of k and ϕ .

The Jacobi zeta function is defined by

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

where $E(m, \phi)$ is the elliptic function of the second kind, $E(m)$ is the complete 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 4379 of file `specfun.h`.

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

Definition at line 4354 of file `specfun.h`.

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

Definition at line 4358 of file `specfun.h`.

8.3.2.201 `float __gnu_cxx::jacobi (unsigned __n, float __alpha, float __beta, float __x) [inline]`

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

See also

[jacobi](#) for details.

Definition at line 2194 of file `specfun.h`.

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

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

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

See also

[jacobi](#) for details.

Definition at line 2205 of file specfun.h.

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

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

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

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

The binomial coefficients are generated by:

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

.

Parameters

<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 4213 of file specfun.h.

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

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

See also

[lbinomial](#) for details.

Definition at line 4184 of file specfun.h.

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

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

See also

[lbinomial](#) for details.

Definition at line 4193 of file `specfun.h`.

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

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

$$n!! = n(n-2)\dots(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)\dots(1), (-1)!! = 1$$

for odd n .

Definition at line 4127 of file `specfun.h`.

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

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

See also

[ldouble_factorial](#) for more details

Definition at line 4100 of file `specfun.h`.

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

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

See also

[double_factorial](#) for more details

Definition at line 4110 of file `specfun.h`.

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

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

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

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

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 4303 of file `specfun.h`.

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

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

See also

[legendre_q](#) for details.

Definition at line 4269 of file `specfun.h`.

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

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

See also

[legendre_q](#) for details.

Definition at line 4279 of file `specfun.h`.

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

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

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

.

Definition at line 4085 of file `specfun.h`.

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

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

See also

[lfactorial](#) for more details

Definition at line 4063 of file `specfun.h`.

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

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

See also

[lfactorial](#) for more details

Definition at line 4073 of file `specfun.h`.

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

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

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

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

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

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

$$\left\{ \begin{matrix} a \\ \nu \end{matrix} \right\}$$

, and others.

Definition at line 3889 of file `specfun.h`.

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

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

See also

[lfalling_factorial](#) for details.

Definition at line 3854 of file `specfun.h`.

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

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

See also

[lfalling_factorial](#) for details.

Definition at line 3864 of file specfun.h.

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

Return the logarithm of the gamma function for real argument.

Definition at line 2915 of file specfun.h.

Referenced by `std::__detail::__laguerre_zeros()`.

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

Return the logarithm of the gamma function for complex argument.

Definition at line 2948 of file specfun.h.

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

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

See also

[lgamma](#) for details.

Definition at line 2897 of file specfun.h.

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

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

See also

[lgamma](#) for details.

Definition at line 2930 of file specfun.h.

8.3.2.222 `long double __gnu_cxx::lgamma(long double __a) [inline]`

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

See also

[lgamma](#) for details.

Definition at line 2907 of file `specfun.h`.

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

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

See also

[lgamma](#) for details.

Definition at line 2940 of file `specfun.h`.

8.3.2.224 `template<typename _Tp> __gnu_cxx::promote_fp_t<_Tp> __gnu_cxx::logint(_Tp __x) [inline]`

Return the logarithmic integral of argument x .

The logarithmic integral is defined by

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

Parameters

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

Definition at line 1688 of file `specfun.h`.

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

Return the logarithmic integral of argument x .

See also

[logint](#) for details.

Definition at line 1664 of file `specfun.h`.

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

Return the logarithmic integral of argument x .

See also

[logint](#) for details.

Definition at line 1673 of file specfun.h.

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

Return the logistic cumulative distribution function.

The formula for the logistic probability function is

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

where $b > 0$.

Definition at line 6463 of file specfun.h.

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

Return the logistic probability density function.

The formula for the logistic probability density function is

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

where $b > 0$.

Definition at line 6446 of file specfun.h.

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

Return the lognormal cumulative probability density function.

The formula for the lognormal cumulative probability density function is

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

Definition at line 6232 of file specfun.h.

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

Return the lognormal probability density function.

The formula for the lognormal probability density function is

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

Definition at line 6215 of file specfun.h.

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

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

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

Thus this function returns

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

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

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

, and others.

Definition at line 3839 of file specfun.h.

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

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

See also

[lrising_factorial](#) for details.

Definition at line 3805 of file specfun.h.

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

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

See also

[lrising_factorial](#) for details.

Definition at line 3815 of file specfun.h.

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

Return the normal cumulative probability density function.

The formula for the normal cumulative probability density function is

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

Definition at line 6199 of file specfun.h.

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

Return the normal probability density function.

The formula for the normal probability density function is

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

Definition at line 6182 of file specfun.h.

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

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

The Owens T function is defined by

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

Parameters

\leftrightarrow _h	The shape factor
\leftrightarrow _a	The integration limit

Definition at line 5771 of file specfun.h.

```
8.3.2.237  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 5743 of file specfun.h.

8.3.2.238 `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 5753 of file specfun.h.

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

Definition at line 4324 of file specfun.h.

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

Definition at line 4312 of file specfun.h.

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

Definition at line 4316 of file specfun.h.

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

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

The polylogarithm function is defined by

Parameters

$_s$	
$_w$	

Definition at line 4970 of file specfun.h.

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

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

The polylogarithm function is defined by

Parameters

\leftarrow __s	
\leftarrow __w	

Definition at line 5010 of file specfun.h.

```
8.3.2.244  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 4943 of file specfun.h.

```
8.3.2.245  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 4983 of file specfun.h.

```
8.3.2.246  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 4953 of file specfun.h.

8.3.2.247 `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 4993 of file specfun.h.

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

Return the psi or digamma function of argument x .

The the psi or digamma function is defined by

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

Parameters

<code>__x</code>	The parameter
------------------	---------------

Definition at line 3547 of file specfun.h.

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

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

See also

[psi](#) for details.

Definition at line 3521 of file specfun.h.

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

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

See also

[psi](#) for details.

Definition at line 3531 of file specfun.h.

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

Definition at line 4345 of file specfun.h.

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

Definition at line 4333 of file specfun.h.

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

Definition at line 4337 of file specfun.h.

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

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

The radial polynomials are defined by

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

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

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

See also

[jacobi](#) for details on the Jacobi polynomials.

Template Parameters

<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 2396 of file specfun.h.

8.3.2.255 `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 2357 of file `specfun.h`.

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

8.3.2.256 `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 2368 of file `specfun.h`.

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

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

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

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

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

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

, and others.

Definition at line 3932 of file `specfun.h`.

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

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

See also

[rising_factorial](#) for details.

Definition at line 3904 of file `specfun.h`.

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

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

See also

[rising_factorial](#) for details.

Definition at line 3914 of file `specfun.h`.

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

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

The reperiodized sine function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 5849 of file `specfun.h`.

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

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

See also

[sin_pi](#) for more details.

Definition at line 5822 of file `specfun.h`.

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

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

See also

[sin_pi](#) for more details.

Definition at line 5832 of file `specfun.h`.

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

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

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

Template Parameters

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

Parameters

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

Definition at line 1609 of file `specfun.h`.

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

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

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

Template Parameters

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

Parameters

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

Definition at line 1650 of file `specfun.h`.

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

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

See also

[sinc](#) for details.

Definition at line 1624 of file `specfun.h`.

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

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

See also

[sinc](#) for details.

Definition at line 1634 of file specfun.h.

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

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

See also

[sinc_pi](#) for details.

Definition at line 1583 of file specfun.h.

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

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

See also

[sinc_pi](#) for details.

Definition at line 1593 of file specfun.h.

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

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

See also

[sincos](#) for details.

Definition at line 6087 of file specfun.h.

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

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

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

Definition at line 6098 of file `specfun.h`.

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

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

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

Definition at line 6132 of file `specfun.h`.

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

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

See also

[sincos_pi](#) for details.

Definition at line 6110 of file `specfun.h`.

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

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

See also

[sincos_pi](#) for details.

Definition at line 6120 of file `specfun.h`.

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

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

Definition at line 6069 of file `specfun.h`.

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

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

See also

[sincos](#) for details.

Definition at line 6078 of file `specfun.h`.

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

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

The reperiodized hyperbolic sine function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 5891 of file `specfun.h`.

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

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

See also

[sinh_pi](#) for more details.

Definition at line 5864 of file `specfun.h`.

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

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

See also

[sinh_pi](#) for more details.

Definition at line 5874 of file `specfun.h`.

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

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

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

Template Parameters

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

Parameters

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

Definition at line 2478 of file `specfun.h`.

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

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

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

Template Parameters

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

Parameters

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

Definition at line 2437 of file `specfun.h`.

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

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

See also

[sinhc_pi](#) for details.

Definition at line 2411 of file `specfun.h`.

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

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

See also

[sinhc_pi](#) for details.

Definition at line 2421 of file `specfun.h`.

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

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

See also

[sinhc](#) for details.

Definition at line 2452 of file `specfun.h`.

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

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

See also

[sinhc](#) for details.

Definition at line 2462 of file `specfun.h`.

8.3.2.285 `template<typename _Tp> __gnu_cxx::_promote_fp_t<_Tp> __gnu_cxx::sinhint (_Tp __x) [inline]`

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

The hyperbolic sine integral is defined by

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

Template Parameters

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

Parameters

\leftrightarrow	The argument
x	

Definition at line 1808 of file specfun.h.

8.3.2.286 `float __gnu_cxx::sinhintf (float x)` `[inline]`

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

See also

[sinhint](#) for details.

Definition at line 1781 of file specfun.h.

8.3.2.287 `long double __gnu_cxx::sinhintl (long double x)` `[inline]`

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

See also

[sinhint](#) for details.

Definition at line 1791 of file specfun.h.

8.3.2.288 `template<typename Tp > __gnu_cxx::__promote_fp_t< Tp > __gnu_cxx::sinint (Tp x)` `[inline]`

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

The sine integral is defined by

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

Parameters

\leftrightarrow	The real upper integration limit
x	

Definition at line 1727 of file specfun.h.

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

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

See also

[sinint](#) for details.

Definition at line 1702 of file `specfun.h`.

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

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

See also

[sinint](#) for details.

Definition at line 1712 of file `specfun.h`.

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

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

The spherical Bessel function is defined by:

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 2714 of file specfun.h.

8.3.2.292 `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 2685 of file specfun.h.

8.3.2.293 `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 2695 of file specfun.h.

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

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

The spherical Bessel function is defined by:

$$k_n(x) = \left(\frac{\pi}{2x}\right)^{1/2} K_{n+1/2}(x)$$

Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

Parameters

<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 2758 of file `specfun.h`.

8.3.2.295 `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 2729 of file `specfun.h`.

8.3.2.296 `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 2739 of file `specfun.h`.

8.3.2.297 `template<typename _Tp> std::complex<__gnu_cxx::__promote_fp_t<_Tp>> __gnu_cxx::sph_hankel_1(unsigned int __n, _Tp __z) [inline]`

Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$ of nonnegative order `n` and real argument $x \geq 0$.

The spherical Hankel function of the first kind is defined by:

$$h_n^{(1)}(x) = \left(\frac{\pi}{2x}\right)^{1/2} H_{n+1/2}^{(1)}(x)$$

or in terms of the cylindrical Bessel and Neumann functions by:

$$h_n^{(1)}(x) = \left(\frac{\pi}{2x}\right)^{1/2} [J_{n+1/2}(x) + iN_{n+1/2}(x)]$$

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

\leftarrow _n	The non-negative order
\leftarrow _z	The real argument

Definition at line 2622 of file specfun.h.

```
8.3.2.298  template<typename _Tp> std::complex<__gnu_cxx::__promote_fp_t<_Tp>> __gnu_cxx::sph_hankel_1 ( unsigned
            int __n, std::complex<_Tp> __x )  [inline]
```

Return the complex spherical Hankel function of the first kind $h_n^{(1)}(x)$ of non-negative integral n and complex argument x .

The spherical Hankel function of the first kind is defined by

$$h_n^{(1)}(x) = \left(\frac{\pi}{2x}\right)^{1/2} H_{n+1/2}^{(1)}(x) = j_n(x) + in_n(x)$$

where $j_n(x)$ and $n_n(x)$ are the spherical Bessel and Neumann functions respectively.

Parameters

\leftarrow _n	The integral order ≥ 0
\leftarrow _x	The complex argument

Definition at line 4828 of file specfun.h.

```
8.3.2.299  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 2589 of file specfun.h.

```
8.3.2.300  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 4796 of file specfun.h.

8.3.2.301 `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 2599 of file `specfun.h`.

8.3.2.302 `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 4807 of file `specfun.h`.

8.3.2.303 `template<typename _Tp> std::complex<__gnu_cxx::__promote_fp_t<_Tp>> __gnu_cxx::sph_hankel_2(unsigned int __n, _Tp __z) [inline]`

Return the spherical Hankel function of the second kind $h_n^{(2)}(x)$ of nonnegative order n and real argument $x \geq 0$.

The spherical Hankel function of the second kind is defined by:

$$h_n^{(2)}(x) = \left(\frac{\pi}{2x}\right)^{1/2} H_{n+1/2}^{(2)}(x)$$

or in terms of the cylindrical Bessel and Neumann functions by:

$$h_n^{(2)}(x) = \left(\frac{\pi}{2x}\right)^{1/2} [J_{n+1/2}(x) - iN_{n+1/2}(x)]$$

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

<code>__n</code>	The non-negative order
------------------	------------------------

Parameters

\leftrightarrow <code>_z</code>	The real argument
---	-------------------

Definition at line 2670 of file specfun.h.

8.3.2.304 `template<typename _Tp> std::complex<__gnu_cxx::__promote_fp_t<_Tp>> __gnu_cxx::sph_hankel_2(unsigned int __n, std::complex<_Tp> __x) [inline]`

Return the complex spherical Hankel function of the second kind $h_n^{(2)}(x)$ of nonnegative order n and complex argument x .

The spherical Hankel function of the second kind is defined by

$$h_n^{(2)}(x) = \left(\frac{\pi}{2x}\right)^{1/2} H_{n+1/2}^{(2)}(x) = j_n(x) - in_n(x)$$

where $j_n(x)$ and $n_n(x)$ are the spherical Bessel and Neumann functions respectively.

Parameters

\leftrightarrow <code>_n</code>	The integral order ≥ 0
\leftrightarrow <code>_x</code>	The complex argument

Definition at line 4876 of file specfun.h.

8.3.2.305 `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 2637 of file specfun.h.

8.3.2.306 `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 4844 of file specfun.h.

8.3.2.307 `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 2647 of file `specfun.h`.

8.3.2.308 `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 4855 of file `specfun.h`.

8.3.2.309 `template<typename _Ttheta , typename _Tphi > std::complex<__gnu_cxx::__promote_fp_t<_Ttheta, _Tphi> > __gnu_cxx::sph_harmonic (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi) [inline]`

Return the complex spherical harmonic function of degree l , order m , and real zenith angle θ , and azimuth angle ϕ .

The spherical harmonic function is defined by:

$$Y_l^m(\theta, \phi) = (-1)^m \frac{(2l+1)}{4\pi} \frac{(l-m)!}{(l+m)!} P_l^{|m|}(\cos \theta) \exp^{im\phi}$$

Parameters

<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 4928 of file `specfun.h`.

8.3.2.310 `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 4892 of file specfun.h.

```
8.3.2.311  std::complex<long double> __gnu_cxx::sph_harmonicl ( unsigned int __l, int __m, long double __theta, long double
__phi )  [inline]
```

Return the complex spherical harmonic function of degree l , order m , and `long double` zenith angle θ , and azimuth angle ϕ .

See also

[sph_harmonic](#) for details.

Definition at line 4904 of file specfun.h.

```
8.3.2.312  template<typename _Tp> _Tp __gnu_cxx::stirling_1 ( unsigned int __n, unsigned int __m )  [inline]
```

Return the Stirling number of the first kind.

The Stirling numbers of the first kind are the coefficients of the Pochhammer polynomials or the rising factorials:

$$(x)_n = \sum_{k=0}^n \begin{bmatrix} n \\ k \end{bmatrix} x^k$$

The recursion is

$$\begin{bmatrix} n+1 \\ m \end{bmatrix} = \begin{bmatrix} n \\ m-1 \end{bmatrix} - n \begin{bmatrix} n \\ m \end{bmatrix}$$

with starting values

$$\begin{bmatrix} 0 \\ 0 \rightarrow m \end{bmatrix} = 1, 0, 0, \dots, 0$$

and

$$\begin{bmatrix} 0 \rightarrow n \\ 0 \end{bmatrix} = 1, 0, 0, \dots, 0$$

The Stirling number of the first kind is denoted by other symbols in the literature, usually $S_n^{(m)}$.

Todo Develop an iterator model for Stirling numbers of the first kind.

Definition at line 6659 of file specfun.h.

8.3.2.313 `template<typename _Tp> _Tp __gnu_cxx::stirling_2 (unsigned int __n, unsigned int __m) [inline]`

Return the Stirling number of the second kind by series expansion or by recursion.

The series is:

$$\sigma_n^{(m)} = \left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_{k=0}^m \frac{(-1)^{m-k} k^n}{(m-k)! k!}$$

The Stirling number of the second kind is denoted by other symbols in the literature: $\sigma_n^{(m)}$, $S_n^{(m)}$ and others.

Todo Develop an iterator model for Stirling numbers of the second kind.

Definition at line 6682 of file specfun.h.

8.3.2.314 `template<typename _Tt, typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::student_t_cdf (_Tt __t, unsigned int __nu)`

Return the Students T probability function.

The students T propability function is related to the incomplete beta function:

$$A(t|\nu) = 1 - I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right) A(t|\nu) =$$

Parameters

<code>__t</code>	
<code>__nu</code>	

Definition at line 6319 of file specfun.h.

8.3.2.315 `template<typename _Tt, typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::student_t_pdf (_Tt __t, unsigned int __nu)`

Return the complement of the Students T probability function.

The complement of the students T propability function is:

$$A_c(t|\nu) = I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right) = 1 - A(t|\nu)$$

Parameters

<code>__t</code>	
<code>__nu</code>	

Definition at line 6339 of file specfun.h.

8.3.2.316 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::tan_pi (_Tp __x) [inline]`

Return the reperiodized tangent function $\tan_{\pi}(x)$ for real argument x .

The reperiodized tangent function is defined by:

$$\tan_{\pi}(x) = \tan(\pi x)$$

Template Parameters

<code>_Tp</code>	The floating-point type of the argument <code>__x</code> .
------------------	--

Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 6017 of file specfun.h.

8.3.2.317 `float __gnu_cxx::tan_pif (float __x) [inline]`

Return the reperiodized tangent function $\tan_{\pi}(x)$ for `float` argument x .

See also

[tan_pi](#) for more details.

Definition at line 5990 of file specfun.h.

8.3.2.318 `long double __gnu_cxx::tan_pil (long double __x) [inline]`

Return the reperiodized tangent function $\tan_{\pi}(x)$ for `long double` argument x .

See also

[tan_pi](#) for more details.

Definition at line 6000 of file specfun.h.

8.3.2.319 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::tanh_pi (_Tp __x) [inline]`

Return the reperiodized hyperbolic tangent function $\tanh_{\pi}(x)$ for real argument x .

The reperiodized hyperbolic tangent function is defined by:

$$\tanh_{\pi}(x) = \tanh(\pi x)$$

Template Parameters

<code>__Tp</code>	The floating-point type of the argument <code>__x</code> .
-------------------	--

Parameters

<code>__x</code>	The argument
------------------	--------------

Definition at line 6059 of file `specfun.h`.

8.3.2.320 `float __gnu_cxx::tanh_pif (float __x) [inline]`

Return the reperiodized hyperbolic tangent function $\tanh_{\pi}(x)$ for `float` argument x .

See also

[tanh_pi](#) for more details.

Definition at line 6032 of file `specfun.h`.

8.3.2.321 `long double __gnu_cxx::tanh_pil (long double __x) [inline]`

Return the reperiodized hyperbolic tangent function $\tanh_{\pi}(x)$ for `long double` argument x .

See also

[tanh_pi](#) for more details.

Definition at line 6042 of file `specfun.h`.

8.3.2.322 `template<typename _Ta> __gnu_cxx::__promote_fp_t<_Ta> __gnu_cxx::tgamma (_Ta __a) [inline]`

Return the gamma function for real argument.

Definition at line 2980 of file `specfun.h`.

Referenced by `std::__detail::__tricoli_u_naive()`.

8.3.2.323 `template<typename _Ta> std::complex<__gnu_cxx::__promote_fp_t<_Ta>> __gnu_cxx::tgamma (std::complex<_Ta> __a) [inline]`

Return the gamma function for complex argument.

Definition at line 3012 of file `specfun.h`.

```
8.3.2.324 template<typename _Ta, typename _Tp> __gnu_cxx::__promote_fp_t<_Ta, _Tp> __gnu_cxx::tgamma ( _Ta __a, _Tp
__x ) [inline]
```

Return the upper incomplete gamma function $\Gamma(a, x)$. The (upper) incomplete gamma function is defined by

$$\Gamma(a, x) = \int_x^{\infty} t^{a-1} e^{-t} dt$$

Definition at line 3049 of file specfun.h.

```
8.3.2.325 template<typename _Ta, typename _Tp> __gnu_cxx::__promote_fp_t<_Ta, _Tp> __gnu_cxx::tgamma_lower ( _Ta __a,
_Tp __x ) [inline]
```

Return the lower incomplete gamma function $\gamma(a, x)$. The lower incomplete gamma function is defined by

$$\gamma(a, x) = \int_0^x t^{a-1} e^{-t} dt$$

Definition at line 3086 of file specfun.h.

```
8.3.2.326 float __gnu_cxx::tgamma_lowerf ( float __a, float __x ) [inline]
```

Return the lower incomplete gamma function $\gamma(a, x)$ for `float` argument.

See also

[tgamma_lower](#) for details.

Definition at line 3064 of file specfun.h.

```
8.3.2.327 long double __gnu_cxx::tgamma_lowerl ( long double __a, long double __x ) [inline]
```

Return the lower incomplete gamma function $\gamma(a, x)$ for `long double` argument.

See also

[tgamma_lower](#) for details.

Definition at line 3074 of file specfun.h.

8.3.2.328 `float __gnu_cxx::tgammaf (float __a) [inline]`

Return the gamma function for `float` argument.

See also

[lgamma](#) for details.

Definition at line 2962 of file `specfun.h`.

8.3.2.329 `std::complex<float> __gnu_cxx::tgammaf (std::complex< float > __a) [inline]`

Return the gamma function for `std::complex<float>` argument.

See also

[lgamma](#) for details.

Definition at line 2994 of file `specfun.h`.

8.3.2.330 `float __gnu_cxx::tgammaf (float __a, float __x) [inline]`

Return the upper incomplete gamma function $\Gamma(a, x)$ for `float` argument.

See also

[tgamma](#) for details.

Definition at line 3027 of file `specfun.h`.

8.3.2.331 `long double __gnu_cxx::tgammal (long double __a) [inline]`

Return the gamma function for `long double` argument.

See also

[lgamma](#) for details.

Definition at line 2972 of file `specfun.h`.

8.3.2.332 `std::complex<long double> __gnu_cxx::tgammal (std::complex< long double > __a)` `[inline]`

Return the gamma function for `std::complex<long double>` argument.

See also

[lgamma](#) for details.

Definition at line 3004 of file `specfun.h`.

8.3.2.333 `long double __gnu_cxx::tgammal (long double __a, long double __x)` `[inline]`

Return the upper incomplete gamma function $\Gamma(a, x)$ for `long double` argument.

See also

[tgamma](#) for details.

Definition at line 3037 of file `specfun.h`.

8.3.2.334 `template<typename _Tpnu, typename _Tp> __gnu_cxx::__promote_fp_t<_Tpnu, _Tp> __gnu_cxx::theta_1 (_Tpnu __nu, _Tp __x)` `[inline]`

Return the exponential theta-1 function $\theta_1(\nu, x)$ of period ν and argument x .

The Neville theta-1 function is defined by

$$\theta_1(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j - 1/2)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 5369 of file `specfun.h`.

8.3.2.335 `float __gnu_cxx::theta_1f (float __nu, float __x)` `[inline]`

Return the exponential theta-1 function $\theta_1(\nu, x)$ of period ν and argument x .

See also

[theta_1](#) for details.

Definition at line 5341 of file `specfun.h`.

8.3.2.336 `long double __gnu_cxx::theta_1l(long double __nu, long double __x)` `[inline]`

Return the exponential theta-1 function $\theta_1(\nu, x)$ of period ν and argument x .

See also

[theta_1](#) for details.

Definition at line 5351 of file specfun.h.

8.3.2.337 `template<typename _Tpnu, typename _Tp> __gnu_cxx::__promote_fp_t<_Tpnu, _Tp> __gnu_cxx::theta_2(_Tpnu __nu, _Tp __x)` `[inline]`

Return the exponential theta-2 function $\theta_2(\nu, x)$ of period ν and argument x .

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 5412 of file specfun.h.

8.3.2.338 `float __gnu_cxx::theta_2f(float __nu, float __x)` `[inline]`

Return the exponential theta-2 function $\theta_2(\nu, x)$ of period ν and argument x .

See also

[theta_2](#) for details.

Definition at line 5384 of file specfun.h.

8.3.2.339 `long double __gnu_cxx::theta_2l(long double __nu, long double __x)` `[inline]`

Return the exponential theta-2 function $\theta_2(\nu, x)$ of period ν and argument x .

See also

[theta_2](#) for details.

Definition at line 5394 of file specfun.h.

8.3.2.340 `template<typename _Tpnu, typename _Tp> __gnu_cxx::__promote_fp_t<_Tpnu, _Tp> __gnu_cxx::theta_3 (_Tpnu __nu, _Tp __x) [inline]`

Return the exponential theta-3 function $\theta_3(\nu, x)$ of period ν and argument x .

The exponential theta-3 function is defined by

$$\theta_3(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 5455 of file specfun.h.

8.3.2.341 `float __gnu_cxx::theta_3f (float __nu, float __x) [inline]`

Return the exponential theta-3 function $\theta_3(\nu, x)$ of period ν and argument x .

See also

[theta_3](#) for details.

Definition at line 5427 of file specfun.h.

8.3.2.342 `long double __gnu_cxx::theta_3l (long double __nu, long double __x) [inline]`

Return the exponential theta-3 function $\theta_3(\nu, x)$ of period ν and argument x .

See also

[theta_3](#) for details.

Definition at line 5437 of file specfun.h.

8.3.2.343 `template<typename _Tpnu, typename _Tp> __gnu_cxx::__promote_fp_t<_Tpnu, _Tp> __gnu_cxx::theta_4 (_Tpnu __nu, _Tp __x) [inline]`

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period ν and argument x .

The exponential theta-4 function is defined by

$$\theta_4(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j + 1/2)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 5498 of file `specfun.h`.

8.3.2.344 `float __gnu_cxx::theta_4f (float __nu, float __x) [inline]`

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period ν and argument x .

See also

[theta_4](#) for details.

Definition at line 5470 of file `specfun.h`.

8.3.2.345 `long double __gnu_cxx::theta_4l (long double __nu, long double __x) [inline]`

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period ν and argument x .

See also

[theta_4](#) for details.

Definition at line 5480 of file `specfun.h`.

8.3.2.346 `template<typename _Tpk, typename _Tp> __gnu_cxx::__promote_fp_t<_Tpk, _Tp> __gnu_cxx::theta_c (_Tpk __k, _Tp __x) [inline]`

Return the Neville theta-c function $\theta_c(k, x)$ of modulus k and argument x .

The Neville theta-c function is defined by

$$\theta_c(k, x) = \sqrt{\frac{\pi}{2kK(k)}} \theta_1 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

where $q(k)$ is the elliptic nome, $K(k)$ is the complete Legendre elliptic integral of the first kind, and $\theta_1(\nu, x)$ is the exponential theta-1 function.

See also

[ellnome](#), [std::comp_ellint_1](#), and [theta_1](#) for details.

Parameters

\leftrightarrow __k	The modulus $-1 \leq k \leq +1$
\leftrightarrow __x	The argument

Definition at line 5634 of file specfun.h.

8.3.2.347 `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 5602 of file specfun.h.

8.3.2.348 `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 5612 of file specfun.h.

8.3.2.349 `template<typename _Tpk, typename _Tp> __gnu_cxx::__promote_fp_t<_Tpk, _Tp> __gnu_cxx::theta_d (_Tpk __k, _Tp __x) [inline]`

Return the Neville theta-d function $\theta_d(k, x)$ of modulus k and argument x .

The Neville theta-d function is defined by

$$\theta_d(k, x) = \sqrt{\frac{\pi}{2K(k)}} \theta_3 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

where $q(k)$ is the elliptic nome, $K(k)$ is the complete Legendre elliptic integral of the first kind, and $\theta_3(\nu, x)$ is the exponential theta-3 function.

See also

[ellnome](#), [std::comp_ellint_1](#), and [theta_3](#) for details.

Parameters

\leftrightarrow _k	The modulus $-1 \leq k \leq +1$
\leftrightarrow _x	The argument

Definition at line 5681 of file specfun.h.

8.3.2.350 `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 5649 of file specfun.h.

8.3.2.351 `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 5659 of file specfun.h.

8.3.2.352 `template<typename _Tpk, typename _Tp> __gnu_cxx::__promote_fp_t<_Tpk, _Tp> __gnu_cxx::theta_n (_Tpk __k, _Tp __x) [inline]`

Return the Neville theta-n function $\theta_n(k, x)$ of modulus k and argument x .

The Neville theta-n function is defined by

$$\theta_n(k, x) = \sqrt{\frac{\pi}{2k'K(k)}} \theta_4 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

where $q(k)$ is the elliptic nome, $K(k)$ is the complete Legendre elliptic integral of the first kind, and $\theta_4(\nu, x)$ is the exponential theta-4 function.

See also

[ellnome](#), [std::comp_ellint_1](#), and [theta_4](#) for details.

Parameters

\leftrightarrow _k	The modulus $-1 \leq k \leq +1$
\leftrightarrow _x	The argument

Definition at line 5728 of file specfun.h.

8.3.2.353 `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 5696 of file specfun.h.

8.3.2.354 `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 5706 of file specfun.h.

8.3.2.355 `template<typename _Tpk, typename _Tp> __gnu_cxx::__promote_fp_t<_Tpk, _Tp> __gnu_cxx::theta_s (_Tpk __k, _Tp __x) [inline]`

Return the Neville theta-s function $\theta_s(k, x)$ of modulus k and argument x .

The Neville theta-s function is defined by

$$\theta_s(k, x) = \sqrt{\frac{\pi}{2kk'K(k)}} \theta_1 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

where $q(k)$ is the elliptic nome, $K(k)$ is the complete Legendre elliptic integral of the first kind, and $\theta_1(\nu, x)$ is the exponential theta-1 function.

See also

[ellnome](#), [std::comp_ellint_1](#), and [theta_1](#) for details.

Parameters

\leftrightarrow __k	The modulus $-1 \leq k \leq +1$
\leftrightarrow __x	The argument

Definition at line 5587 of file specfun.h.

8.3.2.356 `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 5555 of file specfun.h.

8.3.2.357 `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 5565 of file specfun.h.

8.3.2.358 `template<typename _Tpa, typename _Tpc, typename _Tp> __gnu_cxx::__promote_fp_t<_Tpa, _Tpc, _Tp> __gnu_cxx::tricoli_u (_Tpa __a, _Tpc __c, _Tp __x) [inline]`

Return the Tricomi confluent hypergeometric function $U(a, c, x)$ of real numeratorial parameter a , denominatorial parameter c , and argument x .

The Tricomi confluent hypergeometric function is defined by

$$U(a, c, x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a; c; x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1; 2-c; x)$$

where ${}_1F_1(a; c; x)$ if the confluent hypergeometric function.

See also

[conf_hyperg](#).

Parameters

\leftrightarrow _a	The numeratorial parameter
\leftrightarrow _c	The denominatorial parameter
\leftrightarrow _x	The argument

Definition at line 1473 of file specfun.h.

8.3.2.359 `float __gnu_cxx::tricomi_uf (float __a, float __c, float __x) [inline]`

Return the Tricomi confluent hypergeometric function $U(a, c, x)$ of `float` numeratorial parameter a , denominatorial parameter c , and argument x .

See also

[tricomi_u](#) for details.

Definition at line 1439 of file specfun.h.

8.3.2.360 `long double __gnu_cxx::tricomi_ul (long double __a, long double __c, long double __x) [inline]`

Return the Tricomi confluent hypergeometric function $U(a, c, x)$ of `long double` numeratorial parameter a , denominatorial parameter c , and argument x .

See also

[tricomi_u](#) for details.

Definition at line 1450 of file specfun.h.

8.3.2.361 `template<typename _Ta, typename _Tb, typename _Tp> __gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp> __gnu_cxx::weibull_cdf (_Ta __a, _Tb __b, _Tp __x) [inline]`

Return the Weibull cumulative probability density function.

The formula for the Weibull cumulative probability density function is

$$F(x|\lambda) = 1 - e^{-(x/b)^a} \text{ for } x \geq 0$$

Definition at line 6299 of file specfun.h.

```
8.3.2.362  template<typename _Ta , typename _Tb , typename _Tp > __gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp>
           __gnu_cxx::weibull_pdf ( _Ta __a, _Tb __b, _Tp __x )  [inline]
```

Return the Weibull probability density function.

The formula for the Weibull probability density function is

$$f(x|a, b) = \frac{a}{b} \left(\frac{x}{b}\right)^{a-1} \exp - \left(\frac{x}{b}\right)^a \text{ for } x \geq 0$$

Definition at line 6283 of file specfun.h.

```
8.3.2.363  template<typename _Trho , typename _Tphi > __gnu_cxx::__promote_fp_t<_Trho, _Tphi> __gnu_cxx::zernike (
           unsigned int __n, int __m, _Trho __rho, _Tphi __phi )  [inline]
```

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n , signed order m , and real radial argument ρ and azimuthal angle ϕ .

The even Zernicke polynomials are defined by:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

and the odd Zernicke polynomials are defined by:

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative degree m and $m \leq n$ 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 2341 of file specfun.h.

8.3.2.364 `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 2302 of file specfun.h.

8.3.2.365 `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 2313 of file specfun.h.

Chapter 9

Namespace Documentation

9.1 `__gnu_cxx` Namespace Reference

Classes

- struct [__airy_t](#)
- struct [__cyl_bessel_t](#)
- struct [__cyl_hankel_t](#)
- struct [__cyl_mod_bessel_t](#)
- struct [__fock_airy_t](#)
- struct [__fp_is_integer_t](#)
- struct [__gamma_inc_t](#)
- struct [__gamma_temme_t](#)

A structure for the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ 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.

- struct [__jacobi_t](#)
- struct [__lgamma_t](#)
- struct [__pqgamma_t](#)
- struct [__quadrature_point_t](#)
- struct [__sincos_t](#)
- struct [__sph_bessel_t](#)
- struct [__sph_hankel_t](#)
- struct [__sph_mod_bessel_t](#)

Functions

- `template<typename _Tp >`
`bool __fp_is_equal (_Tp __a, _Tp __b, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`__fp_is_integer_t __fp_is_even_integer (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`__fp_is_integer_t __fp_is_half_integer (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`__fp_is_integer_t __fp_is_half_odd_integer (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`__fp_is_integer_t __fp_is_integer (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`__fp_is_integer_t __fp_is_odd_integer (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`bool __fp_is_zero (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`_Tp __fp_max_abs (_Tp __a, _Tp __b)`
- `template<typename _Tp, typename _IntTp >`
`_Tp __parity (_IntTp __k)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t<_Tp> airy_ai (_Tp __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t<_Tp> > airy_ai (std::complex<_Tp> __x)`
- `float airy_aif (float __x)`
- `long double airy_ail (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t<_Tp> airy_bi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t<_Tp> > airy_bi (std::complex<_Tp> __x)`
- `float airy_bif (float __x)`
- `long double airy_bil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t<_Tp> bernoulli (unsigned int __n)`
- `template<typename _Tp >`
`_Tp bernoulli (unsigned int __n, _Tp __x)`
- `float bernoullif (unsigned int __n)`
- `long double bernoullil (unsigned int __n)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t<_Tp> binomial (unsigned int __n, unsigned int __k)`

Return the binomial coefficient as a real number. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t<_Tp> binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)`

Return the binomial cumulative distribution function.

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)`
Return the binomial probability mass function.
- `float binomialf (unsigned int __n, unsigned int __k)`
- `long double binomiall (unsigned int __n, unsigned int __k)`
- `template<typename _Tps, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tps, _Tp > bose_einstein (_Tps __s, _Tp __x)`
- `float bose_einsteinf (float __s, float __x)`
- `long double bose_einsteinl (long double __s, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > chebyshev_t (unsigned int __n, _Tp __x)`
- `float chebyshev_tf (unsigned int __n, float __x)`
- `long double chebyshev_tl (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > chebyshev_u (unsigned int __n, _Tp __x)`
- `float chebyshev_uf (unsigned int __n, float __x)`
- `long double chebyshev_ul (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > chebyshev_v (unsigned int __n, _Tp __x)`
- `float chebyshev_vf (unsigned int __n, float __x)`
- `long double chebyshev_vl (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > chebyshev_w (unsigned int __n, _Tp __x)`
- `float chebyshev_wf (unsigned int __n, float __x)`
- `long double chebyshev_wl (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > clausen (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > clausen (unsigned int __m, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > clausen_cl (unsigned int __m, _Tp __x)`
- `float clausen_clf (unsigned int __m, float __x)`
- `long double clausen_cll (unsigned int __m, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > clausen_sl (unsigned int __m, _Tp __x)`
- `float clausen_slf (unsigned int __m, float __x)`
- `long double clausen_sll (unsigned int __m, long double __x)`
- `float clausenf (unsigned int __m, float __x)`
- `std::complex< float > clausenf (unsigned int __m, std::complex< float > __z)`
- `long double clausenl (unsigned int __m, long double __x)`
- `std::complex< long double > clausenl (unsigned int __m, std::complex< long double > __z)`
- `template<typename _Tk >`
`__gnu_cxx::__promote_fp_t< _Tk > comp_ellint_d (_Tk __k)`
- `float comp_ellint_df (float __k)`
- `long double comp_ellint_dl (long double __k)`
- `float comp_ellint_rf (float __x, float __y)`
- `long double comp_ellint_rl (long double __x, long double __y)`
- `template<typename _Tx, typename _Ty >`
`__gnu_cxx::__promote_fp_t< _Tx, _Ty > comp_ellint_rf (_Tx __x, _Ty __y)`
- `float comp_ellint_rg (float __x, float __y)`
- `long double comp_ellint_rgl (long double __x, long double __y)`

- `template<typename _Tx, typename _Ty >`
`__gnu_cxx::__promote_fp_t< _Tx, _Ty > comp_ellint_rg (_Tx __x, _Ty __y)`
- `template<typename _Tpa, typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > conf_hyperg (_Tpa __a, _Tpc __c, _Tp __x)`
- `template<typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpc, _Tp >::type conf_hyperg_lim (_Tpc __c, _Tp __x)`
- `float conf_hyperg_limf (float __c, float __x)`
- `long double conf_hyperg_liml (long double __c, long double __x)`
- `float conf_hypergf (float __a, float __c, float __x)`
- `long double conf_hypergl (long double __a, long double __c, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > cos_pi (_Tp __x)`
- `float cos_pif (float __x)`
- `long double cos_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > cosh_pi (_Tp __x)`
- `float cosh_pif (float __x)`
- `long double cosh_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > coshint (_Tp __x)`
- `float coshintf (float __x)`
- `long double coshintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > cosint (_Tp __x)`
- `float cosintf (float __x)`
- `long double cosintl (long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > cyl_hankel_1 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > cyl_hankel_1 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > cyl_hankel_1f (float __nu, float __z)`
- `std::complex< float > cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > cyl_hankel_1l (long double __nu, long double __z)`
- `std::complex< long double > cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > cyl_hankel_2 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > > cyl_hankel_2 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > cyl_hankel_2f (float __nu, float __z)`
- `std::complex< float > cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > cyl_hankel_2l (long double __nu, long double __z)`
- `std::complex< long double > cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > dawson (_Tp __x)`
- `float dawsonf (float __x)`
- `long double dawsonl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > debye (unsigned int __n, _Tp __x)`

- float [debyef](#) (unsigned int __n, float __x)
- long double [debyel](#) (unsigned int __n, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [dilog](#) (_Tp __x)
- float [dilogf](#) (float __x)
- long double [dilogl](#) (long double __x)
- template<typename _Tp >
_Tp [dirichlet_beta](#) (_Tp __s)
- float [dirichlet_betaf](#) (float __s)
- long double [dirichlet_betal](#) (long double __s)
- template<typename _Tp >
_Tp [dirichlet_eta](#) (_Tp __s)
- float [dirichlet_etaf](#) (float __s)
- long double [dirichlet_etal](#) (long double __s)
- template<typename _Tp >
_Tp [dirichlet_lambda](#) (_Tp __s)
- float [dirichlet_lambdaf](#) (float __s)
- long double [dirichlet_lambdal](#) (long double __s)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [double_factorial](#) (int __n)

Return the double factorial $n!!$ of the argument as a real number.

$$n!! = n(n-2)\dots(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)\dots(1), (-1)!! = 1$$

for odd n .

- float [double_factorialf](#) (int __n)
- long double [double_factoriall](#) (int __n)
- template<typename _Tk, typename _Tp, typename _Ta, typename _Tb >
__gnu_cxx::__promote_fp_t< _Tk, _Tp, _Ta, _Tb > [ellint_cel](#) (_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)
- float [ellint_celf](#) (float __k_c, float __p, float __a, float __b)
- long double [ellint_cell](#) (long double __k_c, long double __p, long double __a, long double __b)
- template<typename _Tk, typename _Tphi >
__gnu_cxx::__promote_fp_t< _Tk, _Tphi > [ellint_d](#) (_Tk __k, _Tphi __phi)
- float [ellint_df](#) (float __k, float __phi)
- long double [ellint_dl](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Tk >
__gnu_cxx::__promote_fp_t< _Tp, _Tk > [ellint_el1](#) (_Tp __x, _Tk __k_c)
- float [ellint_el1f](#) (float __x, float __k_c)
- long double [ellint_el1l](#) (long double __x, long double __k_c)
- template<typename _Tp, typename _Tk, typename _Ta, typename _Tb >
__gnu_cxx::__promote_fp_t< _Tp, _Tk, _Ta, _Tb > [ellint_el2](#) (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)
- float [ellint_el2f](#) (float __x, float __k_c, float __a, float __b)
- long double [ellint_el2l](#) (long double __x, long double __k_c, long double __a, long double __b)
- template<typename _Tx, typename _Tk, typename _Tp >
__gnu_cxx::__promote_fp_t< _Tx, _Tk, _Tp > [ellint_el3](#) (_Tx __x, _Tk __k_c, _Tp __p)
- float [ellint_el3f](#) (float __x, float __k_c, float __p)
- long double [ellint_el3l](#) (long double __x, long double __k_c, long double __p)
- template<typename _Tp, typename _Up >
__gnu_cxx::__promote_fp_t< _Tp, _Up > [ellint_rc](#) (_Tp __x, _Up __y)
- float [ellint_rcf](#) (float __x, float __y)

- long double [ellint_rcl](#) (long double __x, long double __y)
- template<typename _Tp, typename _Up, typename _Vp >
__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp > [ellint_rd](#) (_Tp __x, _Up __y, _Vp __z)
- float [ellint_rdf](#) (float __x, float __y, float __z)
- long double [ellint_rdl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp > [ellint_rf](#) (_Tp __x, _Up __y, _Vp __z)
- float [ellint_rff](#) (float __x, float __y, float __z)
- long double [ellint_rfl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp > [ellint_rg](#) (_Tp __x, _Up __y, _Vp __z)
- float [ellint_rgf](#) (float __x, float __y, float __z)
- long double [ellint_rgl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp, typename _Wp >
__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp, _Wp > [ellint_rj](#) (_Tp __x, _Up __y, _Vp __z, _Wp __p)
- float [ellint_rjf](#) (float __x, float __y, float __z, float __p)
- long double [ellint_rjl](#) (long double __x, long double __y, long double __z, long double __p)
- template<typename _Tp >
_Tp [ellnome](#) (_Tp __k)
- float [ellnomef](#) (float __k)
- long double [ellnomel](#) (long double __k)
- template<typename _Tp >
_Tp [euler](#) (unsigned int __n)

This returns Euler number E_n .

- template<typename _Tp >
_Tp [eulerian_1](#) (unsigned int __n, unsigned int __m)
- template<typename _Tp >
_Tp [eulerian_2](#) (unsigned int __n, unsigned int __m)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [expint](#) (unsigned int __n, _Tp __x)
- float [expintf](#) (unsigned int __n, float __x)
- long double [expintl](#) (unsigned int __n, long double __x)
- template<typename _Tlam, typename _Tp >
__gnu_cxx::__promote_fp_t< _Tlam, _Tp > [exponential_cdf](#) (_Tlam __lambda, _Tp __x)
- *Return the exponential cumulative probability density function.*
- template<typename _Tlam, typename _Tp >
__gnu_cxx::__promote_fp_t< _Tlam, _Tp > [exponential_pdf](#) (_Tlam __lambda, _Tp __x)
- *Return the exponential probability density function.*
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [factorial](#) (unsigned int __n)

Return the factorial $n!$ of the argument as a real number.

$$n! = 1 \times 2 \times \dots \times n, 0! = 1$$

- float [factorialf](#) (unsigned int __n)
- long double [factoriall](#) (unsigned int __n)
- template<typename _Tp, typename _Tnu >
__gnu_cxx::__promote_fp_t< _Tp, _Tnu > [falling_factorial](#) (_Tp __a, _Tnu __nu)

Return the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n . The falling factorial function is defined by

$$a^{\underline{n}} = \prod_{k=0}^{n-1} (a - k), a^{\underline{0}} = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $n^{\underline{n}} = n!$.

- float [falling_factorialf](#) (float __a, float __nu)
- long double [falling_factoriall](#) (long double __a, long double __nu)
- template<typename _Tps, typename _Tp >
__gnu_cxx::__promote_fp_t< _Tps, _Tp > [fermi_dirac](#) (_Tps __s, _Tp __x)
- float [fermi_diracf](#) (float __s, float __x)
- long double [fermi_diracl](#) (long double __s, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [fisher_f_cdf](#) (_Tp __F, unsigned int __nu1, unsigned int __nu2)
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [fisher_f_pdf](#) (_Tp __F, unsigned int __nu1, unsigned int __nu2)
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [fresnel_c](#) (_Tp __x)
- float [fresnel_cf](#) (float __x)
- long double [fresnel_cl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [fresnel_s](#) (_Tp __x)
- float [fresnel_sf](#) (float __x)
- long double [fresnel_sl](#) (long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > [gamma_cdf](#) (_Ta __alpha, _Tb __beta, _Tp __x)
Return the gamma cumulative propability distribution function.
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > [gamma_pdf](#) (_Ta __alpha, _Tb __beta, _Tp __x)
Return the gamma propability distribution function.
- template<typename _Ta >
__gnu_cxx::__promote_fp_t< _Ta > [gamma_reciprocal](#) (_Ta __a)
- float [gamma_reciprocalf](#) (float __a)
- long double [gamma_reciprocall](#) (long double __a)
- template<typename _Talpha, typename _Tp >
__gnu_cxx::__promote_fp_t< _Talpha, _Tp > [gegenbauer](#) (unsigned int __n, _Talpha __alpha, _Tp __x)
- float [gegenbauerf](#) (unsigned int __n, float __alpha, float __x)
- long double [gegenbauerl](#) (unsigned int __n, long double __alpha, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [harmonic](#) (unsigned int __n)
- template<typename _Tk, typename _Tphi >
__gnu_cxx::__promote_fp_t< _Tk, _Tphi > [heuman_lambda](#) (_Tk __k, _Tphi __phi)
- float [heuman_lambdaf](#) (float __k, float __phi)
- long double [heuman_lambdal](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Up >
__gnu_cxx::__promote_fp_t< _Tp, _Up > [hurwitz_zeta](#) (_Tp __s, _Up __a)

- `template<typename _Tp, typename _Up >`
`std::complex< _Tp > hurwitz_zeta (_Tp __s, std::complex< _Up > __a)`
- `float hurwitz_zetaf (float __s, float __a)`
- `long double hurwitz_zetal (long double __s, long double __a)`
- `template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpa, _Tpb, _Tpc, _Tp > hyperg (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)`
- `float hypergf (float __a, float __b, float __c, float __x)`
- `long double hypergl (long double __a, long double __b, long double __c, long double __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > ibeta (_Ta __a, _Tb __b, _Tp __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > ibetac (_Ta __a, _Tb __b, _Tp __x)`
- `float ibetacf (float __a, float __b, float __x)`
- `long double ibetacld (long double __a, long double __b, long double __x)`
- `float ibetaf (float __a, float __b, float __x)`
- `long double ibetal (long double __a, long double __b, long double __x)`
- `template<typename _Talpha, typename _Tbeta, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Talpha, _Tbeta, _Tp > jacobi (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_fp_t< _Kp, _Up > jacobi_cn (_Kp __k, _Up __u)`
- `float jacobi_cnf (float __k, float __u)`
- `long double jacobi_cnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_fp_t< _Kp, _Up > jacobi_dn (_Kp __k, _Up __u)`
- `float jacobi_dnf (float __k, float __u)`
- `long double jacobi_dnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_fp_t< _Kp, _Up > jacobi_sn (_Kp __k, _Up __u)`
- `float jacobi_snf (float __k, float __u)`
- `long double jacobi_snl (long double __k, long double __u)`
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::__promote_fp_t< _Tk, _Tphi > jacobi_zeta (_Tk __k, _Tphi __phi)`
- `float jacobi_zetaf (float __k, float __phi)`
- `long double jacobi_zetal (long double __k, long double __phi)`
- `float jacobi (unsigned __n, float __alpha, float __beta, float __x)`
- `long double jacobi (unsigned __n, long double __alpha, long double __beta, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > lbinomial (unsigned int __n, unsigned int __k)`

Return the logarithm of the binomial coefficient as a real number. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

- `float lbinomialf (unsigned int __n, unsigned int __k)`
- `long double lbinomial (unsigned int __n, unsigned int __k)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > ldouble_factorial (int __n)`

Return the logarithm of the double factorial $\ln(n!!)$ of the argument as a real number.

$$n!! = n(n-2)\dots(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)\dots(1), (-1)!! = 1$$

for odd n .

- float [ldouble_factorialf](#) (int __n)
- long double [ldouble_factoriall](#) (int __n)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [legendre_q](#) (unsigned int __l, _Tp __x)
- float [legendre_qf](#) (unsigned int __l, float __x)
- long double [legendre_ql](#) (unsigned int __l, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [lfactorial](#) (unsigned int __n)

Return the logarithm of the factorial $\ln(n!)$ of the argument as a real number.

$$n! = 1 \times 2 \times \dots \times n, 0! = 1$$

- float [lfactorialf](#) (unsigned int __n)
- long double [lfactoriall](#) (unsigned int __n)
- template<typename _Tp, typename _Tnu >
__gnu_cxx::__promote_fp_t< _Tp, _Tnu > [lfalling_factorial](#) (_Tp __a, _Tnu __nu)

Return the logarithm of the falling factorial function or the lower Pochhammer symbol. The falling factorial function is defined by

$$a^{\underline{n}} = \Gamma(a+1)/\Gamma(a-n+1) = \prod_{k=0}^{n-1} (a-k), a^{\underline{0}} = 1$$

In particular, $f[n^{\underline{n}}] = n! f$. Thus this function returns

$$\ln[a^{\underline{n}}] = \ln[\Gamma(a+1)] - \ln[\Gamma(a-n+1)], \ln[a^{\underline{0}}] = 0$$

Many notations exist for this function: $(a)_{\nu}$,

$$\left\{ \begin{matrix} a \\ \nu \end{matrix} \right\}$$

, and others.

- float [lfalling_factorialf](#) (float __a, float __nu)
- long double [lfalling_factoriall](#) (long double __a, long double __nu)
- template<typename _Ta >
__gnu_cxx::__promote_fp_t< _Ta > [lgamma](#) (_Ta __a)
- template<typename _Ta >
std::complex< __gnu_cxx::__promote_fp_t< _Ta > > [lgamma](#) (std::complex< _Ta > __a)
- float [lgammaf](#) (float __a)
- std::complex< float > [lgammaf](#) (std::complex< float > __a)
- long double [lgammal](#) (long double __a)
- std::complex< long double > [lgammal](#) (std::complex< long double > __a)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [logint](#) (_Tp __x)
- float [logintf](#) (float __x)
- long double [logintl](#) (long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > [logistic_cdf](#) (_Ta __a, _Tb __b, _Tp __x)

Return the logistic cumulative distribution function.

- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > logistic_pdf (_Ta __a, _Tb __b, _Tp __x)`
Return the logistic probability density function.
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > lognormal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the lognormal cumulative probability density function.
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > lognormal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the lognormal probability density function.
- `template<typename _Tp, typename _Tnu >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tnu > lrising_factorial (_Tp __a, _Tnu __nu)`
Return the logarithm of the rising factorial function or the (upper) Pochhammer symbol. The rising factorial function is defined for integer order by

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(a) = \prod_{k=0}^{\nu-1} (a + k), \overline{0} = 1$$

Thus this function returns

$$\ln[a^{\overline{\nu}}] = \ln[\Gamma(a + \nu)] - \ln[\Gamma(a)], \ln[a^{\overline{0}}] = 0$$

Many notations exist for this function: $(a)_{\nu}$ (especially in the literature of special functions),

$$\left[\begin{array}{c} a \\ \nu \end{array} \right]$$

, and others.

- `float lrising_factorialf (float __a, float __nu)`
- `long double lrising_factoriall (long double __a, long double __nu)`
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > normal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the normal cumulative probability density function.
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > normal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the normal probability density function.
- `template<typename _Tph, typename _Tpa >`
`__gnu_cxx::__promote_fp_t< _Tph, _Tpa > owens_t (_Tph __h, _Tpa __a)`
- `float owens_tf (float __h, float __a)`
- `long double owens_tl (long double __h, long double __a)`
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > pgamma (_Ta __a, _Tp __x)`
- `float pgammaf (float __a, float __x)`
- `long double pgammal (long double __a, long double __x)`
- `template<typename _Tp, typename _Wp >`
`__gnu_cxx::__promote_fp_t< _Tp, _Wp > polylog (_Tp __s, _Wp __w)`
- `template<typename _Tp, typename _Wp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tp, _Wp > > polylog (_Tp __s, std::complex< _Tp > __w)`
- `float polylogf (float __s, float __w)`
- `std::complex< float > polylogf (float __s, std::complex< float > __w)`
- `long double polylogl (long double __s, long double __w)`
- `std::complex< long double > polylogl (long double __s, std::complex< long double > __w)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > psi (_Tp __x)`
- `float psif (float __x)`
- `long double psil (long double __x)`

- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > qgamma (_Ta __a, _Tp __x)`
- `float qgammaf (float __a, float __x)`
- `long double qgamma1 (long double __a, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > radpoly (unsigned int __n, unsigned int __m, _Tp __rho)`
- `float radpolyf (unsigned int __n, unsigned int __m, float __rho)`
- `long double radpoly1 (unsigned int __n, unsigned int __m, long double __rho)`
- `template<typename _Tp, typename _Tnu >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tnu > rising_factorial (_Tp __a, _Tnu __nu)`
Return the rising factorial function or the (upper) Pochhammer function. The rising factorial function is defined by

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(\nu)$$

Many notations exist for this function: $(a)_{\nu}$, (especially in the literature of special functions),

$$\begin{bmatrix} a \\ n \end{bmatrix}$$

, and others.

- `float rising_factorialf (float __a, float __nu)`
- `long double rising_factorial1 (long double __a, long double __nu)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > sin_pi (_Tp __x)`
- `float sin_pif (float __x)`
- `long double sin_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > sinc (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > sinc_pi (_Tp __x)`
- `float sinc_pif (float __x)`
- `long double sinc_pil (long double __x)`
- `float sincf (float __x)`
- `long double sincl (long double __x)`
- `__gnu_cxx::__sincos_t< double > sincos (double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< __gnu_cxx::__promote_fp_t< _Tp > > sincos (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< __gnu_cxx::__promote_fp_t< _Tp > > sincos_pi (_Tp __x)`
- `__gnu_cxx::__sincos_t< float > sincos_pif (float __x)`
- `__gnu_cxx::__sincos_t< long double > sincos_pil (long double __x)`
- `__gnu_cxx::__sincos_t< float > sincosf (float __x)`
- `__gnu_cxx::__sincos_t< long double > sincosl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > sinh_pi (_Tp __x)`
- `float sinh_pif (float __x)`
- `long double sinh_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > sinh_pi (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > sinh_pi (_Tp __x)`
- `float sinh_pif (float __x)`
- `long double sinh_pil (long double __x)`
- `float sinhcf (float __x)`

- long double [sinhcl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [sinhint](#) (_Tp __x)
- float [sinhintf](#) (float __x)
- long double [sinhintl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [sinint](#) (_Tp __x)
- float [sinintf](#) (float __x)
- long double [sinintl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [sph_bessel_i](#) (unsigned int __n, _Tp __x)
- float [sph_bessel_if](#) (unsigned int __n, float __x)
- long double [sph_bessel_il](#) (unsigned int __n, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [sph_bessel_k](#) (unsigned int __n, _Tp __x)
- float [sph_bessel_kf](#) (unsigned int __n, float __x)
- long double [sph_bessel_kl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
std::complex< __gnu_cxx::__promote_fp_t< _Tp > > [sph_hankel_1](#) (unsigned int __n, _Tp __z)
- template<typename _Tp >
std::complex< __gnu_cxx::__promote_fp_t< _Tp > > [sph_hankel_1](#) (unsigned int __n, std::complex< _Tp > __x)
- std::complex< float > [sph_hankel_1f](#) (unsigned int __n, float __z)
- std::complex< float > [sph_hankel_1f](#) (unsigned int __n, std::complex< float > __x)
- std::complex< long double > [sph_hankel_1l](#) (unsigned int __n, long double __z)
- std::complex< long double > [sph_hankel_1l](#) (unsigned int __n, std::complex< long double > __x)
- template<typename _Tp >
std::complex< __gnu_cxx::__promote_fp_t< _Tp > > [sph_hankel_2](#) (unsigned int __n, _Tp __z)
- template<typename _Tp >
std::complex< __gnu_cxx::__promote_fp_t< _Tp > > [sph_hankel_2](#) (unsigned int __n, std::complex< _Tp > __x)
- std::complex< float > [sph_hankel_2f](#) (unsigned int __n, float __z)
- std::complex< float > [sph_hankel_2f](#) (unsigned int __n, std::complex< float > __x)
- std::complex< long double > [sph_hankel_2l](#) (unsigned int __n, long double __z)
- std::complex< long double > [sph_hankel_2l](#) (unsigned int __n, std::complex< long double > __x)
- template<typename _Ttheta, typename _Tphi >
std::complex< __gnu_cxx::__promote_fp_t< _Ttheta, _Tphi > > [sph_harmonic](#) (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)
- std::complex< float > [sph_harmonicf](#) (unsigned int __l, int __m, float __theta, float __phi)
- std::complex< long double > [sph_harmonicl](#) (unsigned int __l, int __m, long double __theta, long double __phi)
- template<typename _Tp >
_Tp [stirling_1](#) (unsigned int __n, unsigned int __m)
- template<typename _Tp >
_Tp [stirling_2](#) (unsigned int __n, unsigned int __m)
- template<typename _Tt, typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [student_t_cdf](#) (_Tt __t, unsigned int __nu)
Return the Students T probability function.
- template<typename _Tt, typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [student_t_pdf](#) (_Tt __t, unsigned int __nu)
Return the complement of the Students T probability function.
- template<typename _Tp >
__gnu_cxx::__promote_fp_t< _Tp > [tan_pi](#) (_Tp __x)

- float [tan_pif](#) (float __x)
- long double [tan_pil](#) (long double __x)
- template<typename _Tp >
 __gnu_cxx::__promote_fp_t< _Tp > [tanh_pi](#) (_Tp __x)
- float [tanh_pif](#) (float __x)
- long double [tanh_pil](#) (long double __x)
- template<typename _Ta >
 __gnu_cxx::__promote_fp_t< _Ta > [tgamma](#) (_Ta __a)
- template<typename _Ta >
 std::complex< __gnu_cxx::__promote_fp_t< _Ta > > [tgamma](#) (std::complex< _Ta > __a)
- template<typename _Ta, typename _Tp >
 __gnu_cxx::__promote_fp_t< _Ta, _Tp > [tgamma](#) (_Ta __a, _Tp __x)
- template<typename _Ta, typename _Tp >
 __gnu_cxx::__promote_fp_t< _Ta, _Tp > [tgamma_lower](#) (_Ta __a, _Tp __x)
- float [tgamma_lowerf](#) (float __a, float __x)
- long double [tgamma_lowerl](#) (long double __a, long double __x)
- float [tgammaf](#) (float __a)
- std::complex< float > [tgammaf](#) (std::complex< float > __a)
- float [tgammaf](#) (float __a, float __x)
- long double [tgamma](#) (long double __a)
- std::complex< long double > [tgamma](#) (std::complex< long double > __a)
- long double [tgamma](#) (long double __a, long double __x)
- template<typename _Tpnu, typename _Tp >
 __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > [theta_1](#) (_Tpnu __nu, _Tp __x)
- float [theta_1f](#) (float __nu, float __x)
- long double [theta_1l](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
 __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > [theta_2](#) (_Tpnu __nu, _Tp __x)
- float [theta_2f](#) (float __nu, float __x)
- long double [theta_2l](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
 __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > [theta_3](#) (_Tpnu __nu, _Tp __x)
- float [theta_3f](#) (float __nu, float __x)
- long double [theta_3l](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
 __gnu_cxx::__promote_fp_t< _Tpnu, _Tp > [theta_4](#) (_Tpnu __nu, _Tp __x)
- float [theta_4f](#) (float __nu, float __x)
- long double [theta_4l](#) (long double __nu, long double __x)
- template<typename _Tpk, typename _Tp >
 __gnu_cxx::__promote_fp_t< _Tpk, _Tp > [theta_c](#) (_Tpk __k, _Tp __x)
- float [theta_cf](#) (float __k, float __x)
- long double [theta_cl](#) (long double __k, long double __x)
- template<typename _Tpk, typename _Tp >
 __gnu_cxx::__promote_fp_t< _Tpk, _Tp > [theta_d](#) (_Tpk __k, _Tp __x)
- float [theta_df](#) (float __k, float __x)
- long double [theta_dl](#) (long double __k, long double __x)
- template<typename _Tpk, typename _Tp >
 __gnu_cxx::__promote_fp_t< _Tpk, _Tp > [theta_n](#) (_Tpk __k, _Tp __x)
- float [theta_nf](#) (float __k, float __x)
- long double [theta_nl](#) (long double __k, long double __x)
- template<typename _Tpk, typename _Tp >
 __gnu_cxx::__promote_fp_t< _Tpk, _Tp > [theta_s](#) (_Tpk __k, _Tp __x)

- float [theta_sf](#) (float __k, float __x)
- long double [theta_sl](#) (long double __k, long double __x)
- template<typename _Tpa, typename _Tpc, typename _Tp >
__gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > [tricomi_u](#) (_Tpa __a, _Tpc __c, _Tp __x)
- float [tricomi_uf](#) (float __a, float __c, float __x)
- long double [tricomi_ul](#) (long double __a, long double __c, long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > [weibull_cdf](#) (_Ta __a, _Tb __b, _Tp __x)
Return the Weibull cumulative probability density function.
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > [weibull_pdf](#) (_Ta __a, _Tb __b, _Tp __x)
Return the Weibull probability density function.
- template<typename _Trho, typename _Tphi >
__gnu_cxx::__promote_fp_t< _Trho, _Tphi > [zernike](#) (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)
- float [zernikef](#) (unsigned int __n, int __m, float __rho, float __phi)
- long double [zernikel](#) (unsigned int __n, int __m, long double __rho, long double __phi)

9.1.1 Function Documentation

9.1.1.1 `template<typename _Tp> bool __gnu_cxx::__fp_is_equal (_Tp __a, _Tp __b, _Tp __mul = _Tp{1}) [inline]`

A function to reliably compare two floating point numbers.

Parameters

<code>__a</code>	The left hand side
<code>__b</code>	The right hand side
<code>__mul</code>	The multiplier for numeric epsilon for comparison

Returns

`true` if `a` and `b` are equal to zero or differ only by $\max(a, b) * \text{mul} * \text{epsilon}$

Definition at line 81 of file `math_util.h`.

References `__fp_max_abs()`.

Referenced by `__fp_is_half_integer()`, `__fp_is_half_odd_integer()`, `__fp_is_integer()`, `std::__detail::__polylog()`, `std::__detail::__polylog_exp_neg()`, `std::__detail::__polylog_exp_neg_int()`, `std::__detail::__polylog_exp_pos_int()`, and `std::__detail::__polylog_exp_pos_real()`.

9.1.1.2 `template<typename _Tp> __fp_is_integer_t __gnu_cxx::__fp_is_even_integer (_Tp __a, _Tp __mul = _Tp{1}) [inline]`

A function to reliably detect if a floating point number is an even integer.

Parameters

<code>__a</code>	The floating point number
<code>__mul</code>	The multiplier of machine epsilon for the tolerance

Returns

`true` if `a` is an even integer within `mul * epsilon`.

Definition at line 217 of file `math_util.h`.

References `__fp_is_integer()`.

Referenced by `std::__detail::__riemann_zeta_glob()`.

```
9.1.1.3 template<typename _Tp> __fp_is_integer_t __gnu_cxx::__fp_is_half_integer ( _Tp __a, _Tp __mul = _Tp{1} )
        [inline]
```

A function to reliably detect if a floating point number is a half-integer.

Parameters

<code>__a</code>	The floating point number
<code>__mul</code>	The multiplier of machine epsilon for the tolerance

Returns

`true` if `2a` is an integer within `mul * epsilon` and the returned value is half the integer, `int(a) / 2`.

Definition at line 172 of file `math_util.h`.

References `__fp_is_equal()`.

```
9.1.1.4 template<typename _Tp> __fp_is_integer_t __gnu_cxx::__fp_is_half_odd_integer ( _Tp __a, _Tp __mul = _Tp{1} )
        [inline]
```

A function to reliably detect if a floating point number is a half-odd-integer.

Parameters

<code>__a</code>	The floating point number
<code>__mul</code>	The multiplier of machine epsilon for the tolerance

Returns

`true` if `2a` is an odd integer within `mul * epsilon` and the returned value is `int(a - 1) / 2`.

Definition at line 195 of file `math_util.h`.

References `__fp_is_equal()`.

Referenced by `std::__detail::__psi()`.

```
9.1.1.5  template<typename _Tp> __fp_is_integer_t __gnu_cxx::__fp_is_integer ( _Tp __a, _Tp __mul = _Tp{1} )
        [inline]
```

A function to reliably detect if a floating point number is an integer.

Parameters

<code>__a</code>	The floating point number
<code>__mul</code>	The multiplier of machine epsilon for the tolerance

Returns

`true` if `a` is an integer within `mul * epsilon`.

Definition at line 150 of file `math_util.h`.

References `__fp_is_equal()`.

Referenced by `std::__detail::__dirichlet_eta()`, `std::__detail::__falling_factorial()`, `__fp_is_even_integer()`, `__fp_is_↵
odd_integer()`, `std::__detail::__gamma()`, `std::__detail::__gamma_reciprocal()`, `std::__detail::__gamma_series()`, `std↵
::__detail::__log_falling_factorial()`, `std::__detail::__log_gamma()`, `std::__detail::__pgamma()`, `std::__detail::__polylog()`,
`std::__detail::__polylog_exp()`, `std::__detail::__psi()`, `std::__detail::__qgamma()`, `std::__detail::__riemann_zeta()`, `std↵
::__detail::__riemann_zeta_m_1()`, `std::__detail::__tgamma()`, `std::__detail::__tgamma_lower()`, and `std::__detail::__↵
tricoli_u_naive()`.

```
9.1.1.6  template<typename _Tp> __fp_is_integer_t __gnu_cxx::__fp_is_odd_integer ( _Tp __a, _Tp __mul = _Tp{1} )
        [inline]
```

A function to reliably detect if a floating point number is an odd integer.

Parameters

<code>__a</code>	The floating point number
<code>__mul</code>	The multiplier of machine epsilon for the tolerance

Returns

`true` if `a` is an odd integer within `mul * epsilon`.

Definition at line 237 of file `math_util.h`.

References `__fp_is_integer()`.

9.1.1.7 `template<typename _Tp> bool __gnu_cxx::__fp_is_zero (_Tp __a, _Tp __mul = _Tp{1}) [inline]`

A function to reliably compare a floating point number with zero.

Parameters

<code>__a</code>	The floating point number
<code>__mul</code>	The multiplier for numeric epsilon for comparison

Returns

`true` if `a` and `b` are equal to zero or differ only by $\max(a, b) * mul * epsilon$

Definition at line 106 of file `math_util.h`.

Referenced by `std::__detail::__polylog()`, `std::__detail::__polylog_exp_neg()`, `std::__detail::__polylog_exp_neg_int()`, `std::__detail::__polylog_exp_pos_int()`, and `std::__detail::__polylog_exp_pos_real()`.

9.1.1.8 `template<typename _Tp> _Tp __gnu_cxx::__fp_max_abs (_Tp __a, _Tp __b) [inline]`

A function to return the max of the absolute values of two numbers ... so we won't include everything.

Parameters

<code>__↵ __a</code>	The left hand side
<code>__↵ __b</code>	The right hand side

Definition at line 58 of file `math_util.h`.

Referenced by `__fp_is_equal()`.

9.1.1.9 `template<typename _Tp, typename _IntTp> _Tp __gnu_cxx::__parity (_IntTp __k) [inline]`

Return -1 if the integer argument is odd and +1 if it is even.

Definition at line 47 of file `math_util.h`.

Referenced by `std::__detail::__stirling_1_series()`.

9.2 std Namespace Reference

Namespaces

- [__detail](#)

Functions

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x)`
- `float assoc_laguerref (unsigned int __n, unsigned int __m, float __x)`
- `long double assoc_laguerrel (unsigned int __n, unsigned int __m, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > assoc_legendre (unsigned int __l, unsigned int __m, _Tp __x)`
- `float assoc_legendref (unsigned int __l, unsigned int __m, float __x)`
- `long double assoc_legendrel (unsigned int __l, unsigned int __m, long double __x)`
- `template<typename _Tpa, typename _Tpb >`
`__gnu_cxx::__promote_fp_t< _Tpa, _Tpb > beta (_Tpa __a, _Tpb __b)`
- `float betaf (float __a, float __b)`
- `long double betal (long double __a, long double __b)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > comp_ellint_1 (_Tp __k)`
- `float comp_ellint_1f (float __k)`
- `long double comp_ellint_1l (long double __k)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > comp_ellint_2 (_Tp __k)`
- `float comp_ellint_2f (float __k)`
- `long double comp_ellint_2l (long double __k)`
- `template<typename _Tp, typename _Tpn >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tpn > comp_ellint_3 (_Tp __k, _Tpn __nu)`
- `float comp_ellint_3f (float __k, float __nu)`
Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for float modulus k.
- `long double comp_ellint_3l (long double __k, long double __nu)`
Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for long double modulus k.
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > cyl_bessel_i (_Tpnu __nu, _Tp __x)`
- `float cyl_bessel_if (float __nu, float __x)`
- `long double cyl_bessel_il (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > cyl_bessel_j (_Tpnu __nu, _Tp __x)`
- `float cyl_bessel_jf (float __nu, float __x)`
- `long double cyl_bessel_jl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > cyl_bessel_k (_Tpnu __nu, _Tp __x)`
- `float cyl_bessel_kf (float __nu, float __x)`
- `long double cyl_bessel_kl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > cyl_neumann (_Tpnu __nu, _Tp __x)`
- `float cyl_neumannf (float __nu, float __x)`

- long double [cyl_neumannl](#) (long double __nu, long double __x)
- template<typename _Tp, typename _Tpp >
[__gnu_cxx::__promote_fp_t](#)< _Tp, _Tpp > [ellint_1](#) (_Tp __k, _Tpp __phi)
- float [ellint_1f](#) (float __k, float __phi)
- long double [ellint_1l](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Tpp >
[__gnu_cxx::__promote_fp_t](#)< _Tp, _Tpp > [ellint_2](#) (_Tp __k, _Tpp __phi)
- float [ellint_2f](#) (float __k, float __phi)
- Return the incomplete elliptic integral of the second kind $E(k, \phi)$ for float argument.*
- long double [ellint_2l](#) (long double __k, long double __phi)
- Return the incomplete elliptic integral of the second kind $E(k, \phi)$.*
- template<typename _Tp, typename _Tpn, typename _Tpp >
[__gnu_cxx::__promote_fp_t](#)< _Tp, _Tpn, _Tpp > [ellint_3](#) (_Tp __k, _Tpn __nu, _Tpp __phi)
- Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.*
- float [ellint_3f](#) (float __k, float __nu, float __phi)
- Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ for float argument.*
- long double [ellint_3l](#) (long double __k, long double __nu, long double __phi)
- Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.*
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [expint](#) (_Tp __x)
- float [expintf](#) (float __x)
- long double [expintl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [hermite](#) (unsigned int __n, _Tp __x)
- float [hermitef](#) (unsigned int __n, float __x)
- long double [hermitel](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [laguerre](#) (unsigned int __n, _Tp __x)
- float [laguerref](#) (unsigned int __n, float __x)
- long double [laguerrel](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [legendre](#) (unsigned int __l, _Tp __x)
- float [legendref](#) (unsigned int __l, float __x)
- long double [legendrel](#) (unsigned int __l, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [riemann_zeta](#) (_Tp __s)
- float [riemann_zetaf](#) (float __s)
- long double [riemann_zetal](#) (long double __s)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [sph_bessel](#) (unsigned int __n, _Tp __x)
- float [sph_besself](#) (unsigned int __n, float __x)
- long double [sph_bessell](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [sph_legendre](#) (unsigned int __l, unsigned int __m, _Tp __theta)
- float [sph_legendref](#) (unsigned int __l, unsigned int __m, float __theta)
- long double [sph_legendrel](#) (unsigned int __l, unsigned int __m, long double __theta)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [sph_neumann](#) (unsigned int __n, _Tp __x)
- float [sph_neumannf](#) (unsigned int __n, float __x)
- long double [sph_neumannl](#) (unsigned int __n, long double __x)

9.3 std::__detail Namespace Reference

Classes

- struct [__gamma_lanczos_data](#)
- struct [__gamma_lanczos_data< double >](#)
- struct [__gamma_lanczos_data< float >](#)
- struct [__gamma_lanczos_data< long double >](#)
- struct [__gamma_spouge_data](#)
- struct [__gamma_spouge_data< double >](#)
- struct [__gamma_spouge_data< float >](#)
- struct [__gamma_spouge_data< long double >](#)
- class [_Airy](#)
- class [_Airy_asymp](#)
- struct [_Airy_asymp_data](#)
- struct [_Airy_asymp_data< double >](#)
- struct [_Airy_asymp_data< float >](#)
- struct [_Airy_asymp_data< long double >](#)
- class [_Airy_asymp_series](#)
- struct [_Airy_default_radii](#)
- struct [_Airy_default_radii< double >](#)
- struct [_Airy_default_radii< float >](#)
- struct [_Airy_default_radii< long double >](#)
- class [_Airy_series](#)
- struct [_AiryAuxilliaryState](#)
- struct [_AiryState](#)
- class [_AsympTerminator](#)
- struct [_Factorial_table](#)
- class [_Terminator](#)

Functions

- template<typename _Tp >
[__gnu_cxx::__airy_t](#)< _Tp, _Tp > [__airy](#) (_Tp __z)
Compute the Airy functions $Ai(x)$ and $Bi(x)$ and their first derivatives $Ai'(x)$ and $Bi'(x)$ respectively.
- template<typename _Tp >
std::complex< _Tp > [__airy_ai](#) (std::complex< _Tp > __z)
Return the complex Airy Ai function.
- template<typename _Tp >
void [__airy_arg](#) (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)
Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.
- template<typename _Tp >
std::complex< _Tp > [__airy_bi](#) (std::complex< _Tp > __z)
Return the complex Airy Bi function.
- template<typename _Tp >
_Tp [__assoc_laguerre](#) (unsigned int __n, unsigned int __m, _Tp __x)
This routine returns the associated Laguerre polynomial of order n, degree m: $L_n^m(x)$.

- template<typename _Tp >
_Tp [__assoc_legendre_p](#) (unsigned int __l, unsigned int __m, _Tp __x)
Return the associated Legendre function by recursion on l and downward recursion on m.
- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp [__bernoulli](#) (unsigned int __n)
This returns Bernoulli number B_n .
- template<typename _Tp >
_Tp [__bernoulli](#) (unsigned int __n, _Tp __x)
- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp [__bernoulli_2n](#) (unsigned int __n)
This returns Bernoulli number B_{2n} at even integer arguments $2n$.
- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp [__bernoulli_series](#) (unsigned int __n)
This returns Bernoulli numbers from a table or by summation for larger values.

$$B_{2n} = (-1)^{n+1} 2 \frac{(2n)!}{(2\pi)^{2n}} \zeta(2n)$$

- template<typename _Tp >
_Tp [__beta](#) (_Tp __a, _Tp __b)
Return the beta function $B(a, b)$.
- template<typename _Tp >
_Tp [__beta_gamma](#) (_Tp __a, _Tp __b)
Return the beta function: $B(a, b)$.
- template<typename _Tp >
_Tp [__beta_inc](#) (_Tp __a, _Tp __b, _Tp __x)
- template<typename _Tp >
_Tp [__beta_lgamma](#) (_Tp __a, _Tp __b)
Return the beta function $B(a, b)$ using the log gamma functions.
- template<typename _Tp >
_Tp [__beta_product](#) (_Tp __a, _Tp __b)
Return the beta function $B(x, y)$ using the product form.
- template<typename _Tp >
_Tp [__binomial](#) (unsigned int __n, unsigned int __k)
Return the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

- template<typename _Tp >
_Tp [__binomial](#) (_Tp __nu, unsigned int __k)
Return the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

- `template<typename _Tp >`
`_Tp __binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)`
Return the binomial cumulative distribution function.
- `template<typename _Tp >`
`_Tp __binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)`
Return the complementary binomial cumulative distribution function.
- `template<typename _Tp >`
`_Tp __binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)`
Return the binomial probability mass function.
- `template<typename _Sp, typename _Tp >`
`_Tp __bose_einstein (_Sp __s, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_recur (unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1)`
- `template<typename _Tp >`
`_Tp __chebyshev_t (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_u (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_v (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_w (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chi_squared_pdf (_Tp __chi2, unsigned int __nu)`
Return the chi-squared probability function. This returns the probability that the observed chi-squared for a correct model is less than the value χ^2 .
- `template<typename _Tp >`
`_Tp __chi_squared_pdfc (_Tp __chi2, unsigned int __nu)`
Return the complementary chi-squared probability function. This returns the probability that the observed chi-squared for a correct model is greater than the value χ^2 .
- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __chshint (_Tp __x, _Tp &_Chi, _Tp &_Shi)`
This function returns the hyperbolic cosine $Ci(x)$ and hyperbolic sine $Si(x)$ integrals as a pair.
- `template<typename _Tp >`
`void __chshint_cont_frac (_Tp __t, _Tp &_Chi, _Tp &_Shi)`
This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by continued fraction for positive argument.
- `template<typename _Tp >`
`void __chshint_series (_Tp __t, _Tp &_Chi, _Tp &_Shi)`
This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by series summation for positive argument.
- `template<typename _Tp >`
`std::complex< _Tp > __clamp_0_m2pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`std::complex< _Tp > __clamp_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`std::complex< _Tp > __clausen (unsigned int __m, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp __clausen (unsigned int __m, _Tp __x)`
- `template<typename _Tp >`
`_Tp __clausen_cl (unsigned int __m, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp __clausen_cl (unsigned int __m, _Tp __x)`

- `template<typename _Tp >`
`_Tp __clausen_sl (unsigned int __m, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp __clausen_sl (unsigned int __m, _Tp __x)`
- `template<typename _Tp >`
`_Tp __comp_ellint_1 (_Tp __k)`
Return the complete elliptic integral of the first kind $K(k)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __comp_ellint_2 (_Tp __k)`
Return the complete elliptic integral of the second kind $E(k)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __comp_ellint_3 (_Tp __k, _Tp __nu)`
Return the complete elliptic integral of the third kind $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __comp_ellint_d (_Tp __k)`
- `template<typename _Tp >`
`_Tp __comp_ellint_rf (_Tp __x, _Tp __y)`
- `template<typename _Tp >`
`_Tp __comp_ellint_rg (_Tp __x, _Tp __y)`
- `template<typename _Tp >`
`_Tp __conf_hyperg (_Tp __a, _Tp __c, _Tp __x)`
Return the confluent hypergeometric function ${}_1F_1(a; c; x) = M(a, c, x)$.
- `template<typename _Tp >`
`_Tp __conf_hyperg_lim (_Tp __c, _Tp __x)`
Return the confluent hypergeometric limit function ${}_0F_1(-; c; x)$.
- `template<typename _Tp >`
`_Tp __conf_hyperg_lim_series (_Tp __c, _Tp __x)`
This routine returns the confluent hypergeometric limit function by series expansion.
- `template<typename _Tp >`
`_Tp __conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)`
Return the hypergeometric function ${}_1F_1(a; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.
- `template<typename _Tp >`
`_Tp __conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)`
This routine returns the confluent hypergeometric function by series expansion.
- `template<typename _Tp >`
`_Tp __cos_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __cos_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp __cosh_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __cosh_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp __coshint (const _Tp __x)`
Return the hyperbolic cosine integral $Chi(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Bessel function.
- `template<typename _Tp >`
`_Tp __cyl_bessel_i (_Tp __nu, _Tp __x)`

Return the regular modified Bessel function of order ν : $I_\nu(x)$.

- `template<typename _Tp >`
`_Tp __cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)`
 This routine returns the cylindrical Bessel functions of order ν : J_ν or I_ν by series expansion.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_ik (_Tp __nu, _Tp __x)`
 Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_ik_asymp (_Tp __nu, _Tp __x)`
 This routine computes the asymptotic modified cylindrical Bessel and functions of order ν : $I_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_ik_steep (_Tp __nu, _Tp __x)`
 Compute the modified Bessel functions $I_\nu(x)$ and $K_\nu(x)$ and their first derivatives $I'_\nu(x)$ and $K'_\nu(x)$ respectively. These four functions are computed together for numerical stability.
- `template<typename _Tp >`
`_Tp __cyl_bessel_j (_Tp __nu, _Tp __x)`
 Return the Bessel function of order ν : $J_\nu(x)$.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_jn (_Tp __nu, _Tp __x)`
 Return the cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_jn_asymp (_Tp __nu, _Tp __x)`
 This routine computes the asymptotic cylindrical Bessel and Neumann functions of order ν : $J_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_bessel_t< _Tp, _Tp, std::complex< _Tp > > __cyl_bessel_jn_neg_arg (_Tp __nu, _Tp __x)`
 Return the cylindrical Bessel functions and their derivatives of order ν and argument $x < 0$.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_jn_steep (_Tp __nu, _Tp __x)`
 Compute the Bessel $J_\nu(x)$ and Neumann $N_\nu(x)$ functions and their first derivatives $J'_\nu(x)$ and $N'_\nu(x)$ respectively. These four functions are computed together for numerical stability.
- `template<typename _Tp >`
`_Tp __cyl_bessel_k (_Tp __nu, _Tp __x)`
 Return the irregular modified Bessel function $K_\nu(x)$ of order ν .
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_hankel_1 (_Tp __nu, _Tp __x)`
 Return the cylindrical Hankel function of the first kind $H_\nu^{(1)}(x)$.
- `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 $H_n^{(2)}(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_hankel_2 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
 Return the complex cylindrical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_neumann (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
 Return the complex cylindrical Neumann function.

- `template<typename _Tp >`
`_Tp __cyl_neumann_n (_Tp __nu, _Tp __x)`
Return the Neumann function of order ν : $N_\nu(x)$.
- `template<typename _Tp >`
`_Tp __dawson (_Tp __x)`
Return the Dawson integral, $F(x)$, for real argument x .
- `template<typename _Tp >`
`_Tp __dawson_cont_frac (_Tp __x)`
Compute the Dawson integral using a sampling theorem representation.
- `template<typename _Tp >`
`_Tp __dawson_series (_Tp __x)`
Compute the Dawson integral using the series expansion.
- `template<typename _Tp >`
`_Tp __debye (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`void __debye_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)`
- `template<typename _Tp >`
`_Tp __dilog (_Tp __x)`
Compute the dilogarithm function $Li_2(x)$ by summation for $x \leq 1$.
- `template<typename _Tp >`
`_Tp __dirichlet_beta (std::complex< _Tp > __s)`
- `template<typename _Tp >`
`_Tp __dirichlet_beta (_Tp __s)`
- `template<typename _Tp >`
`std::complex< _Tp > __dirichlet_eta (std::complex< _Tp > __s)`
- `template<typename _Tp >`
`_Tp __dirichlet_eta (_Tp __s)`
- `template<typename _Tp >`
`_Tp __dirichlet_lambda (_Tp __s)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __double_factorial (int __n)`
Return the double factorial of the integer n .
- `template<typename _Tp >`
`_Tp __ellint_1 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_2 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the second kind $E(k, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)`
Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp __ellint_d (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`
`_Tp __ellint_el1 (_Tp __x, _Tp __k_c)`
- `template<typename _Tp >`
`_Tp __ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp __ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`

- `template<typename _Tp >`
`_Tp __ellint_rc (_Tp __x, _Tp __y)`
Return the Carlson elliptic function $R_C(x, y) = R_F(x, y, y)$ where $R_F(x, y, z)$ is the Carlson elliptic function of the first kind.
- `template<typename _Tp >`
`_Tp __ellint_rd (_Tp __x, _Tp __y, _Tp __z)`
Return the Carlson elliptic function of the second kind $R_D(x, y, z) = R_J(x, y, z, z)$ where $R_J(x, y, z, p)$ is the Carlson elliptic function of the third kind.
- `template<typename _Tp >`
`_Tp __ellint_rf (_Tp __x, _Tp __y, _Tp __z)`
Return the Carlson elliptic function $R_F(x, y, z)$ of the first kind.
- `template<typename _Tp >`
`_Tp __ellint_rg (_Tp __x, _Tp __y, _Tp __z)`
Return the symmetric Carlson elliptic function of the second kind $R_G(x, y, z)$.
- `template<typename _Tp >`
`_Tp __ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)`
Return the Carlson elliptic function $R_J(x, y, z, p)$ of the third kind.
- `template<typename _Tp >`
`_Tp __ellnome (_Tp __k)`
- `template<typename _Tp >`
`_Tp __ellnome_k (_Tp __k)`
- `template<typename _Tp >`
`_Tp __ellnome_series (_Tp __k)`
- `template<typename _Tp >`
`_Tp __euler (unsigned int __n)`
This returns Euler number E_n .
- `template<typename _Tp >`
`_Tp __euler (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp __euler_series (unsigned int __n)`
- `template<typename _Tp >`
`_Tp __eulerian_1_recur (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __eulerian_2 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __eulerian_2_recur (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __expint (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$.
- `template<typename _Tp >`
`_Tp __expint (_Tp __x)`
Return the exponential integral $Ei(x)$.
- `template<typename _Tp >`
`_Tp __expint_E1 (_Tp __x)`
Return the exponential integral $E_1(x)$.
- `template<typename _Tp >`
`_Tp __expint_E1_asymp (_Tp __x)`
Return the exponential integral $E_1(x)$ by asymptotic expansion.
- `template<typename _Tp >`
`_Tp __expint_E1_series (_Tp __x)`

Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.

- `template<typename _Tp >`
`_Tp __expint_Ei (_Tp __x)`

Return the exponential integral $Ei(x)$.

- `template<typename _Tp >`
`_Tp __expint_Ei_asymp (_Tp __x)`

Return the exponential integral $Ei(x)$ by asymptotic expansion.

- `template<typename _Tp >`
`_Tp __expint_Ei_series (_Tp __x)`

Return the exponential integral $Ei(x)$ by series summation.

- `template<typename _Tp >`
`_Tp __expint_En_asymp (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ for large argument.

- `template<typename _Tp >`
`_Tp __expint_En_cont_frac (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by continued fractions.

- `template<typename _Tp >`
`_Tp __expint_En_large_n (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ for large order.

- `template<typename _Tp >`
`_Tp __expint_En_recursion (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.

- `template<typename _Tp >`
`_Tp __expint_En_series (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by series summation.

- `template<typename _Tp >`
`_Tp __exponential_cdf (_Tp __lambda, _Tp __x)`

Return the exponential cumulative probability density function.

- `template<typename _Tp >`
`_Tp __exponential_cdfc (_Tp __lambda, _Tp __x)`

Return the complement of the exponential cumulative probability density function.

- `template<typename _Tp >`
`_Tp __exponential_pdf (_Tp __lambda, _Tp __x)`

Return the exponential probability density function.

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __factorial (unsigned int __n)`

Return the factorial of the integer n .

- `template<typename _Tp >`
`_Tp __falling_factorial (_Tp __a, int __n)`

Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n . The falling factorial function is defined by

$$a^{\underline{n}} = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $n^{\underline{n}} = n!$.

- `template<typename _Tp >`
`_Tp __falling_factorial (_Tp __a, _Tp __nu)`

Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and order ν . The falling factorial function is defined by

$$a^{\underline{\nu}} = \Gamma(a+1)/\Gamma(a-\nu+1)$$

- `template<typename _Sp, typename _Tp >`
`_Tp __fermi_dirac (_Sp __s, _Tp __x)`

- `template<typename _Tp >`
`_Tp __fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`

Return the F -distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .

- `template<typename _Tp >`
`_Tp __fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)`

Return the F -distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .

- `template<typename _Tp >`
`_Tp __fisher_f_pdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`

Return the F -distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .

- `template<typename _Tp >`
`__gnu_cxx::__fock_airy_t< _Tp, std::complex< _Tp > > __fock_airy (_Tp __x)`

Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w'_1(x)$ and $w'_2(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$

- `template<typename _Tp >`
`std::complex< _Tp > __fresnel (const _Tp __x)`

Return the Fresnel cosine and sine integrals as a complex number $\$f[C(x) + iS(x)]$.

- `template<typename _Tp >`
`void __fresnel_cont_frac (const _Tp __ax, _Tp &__Cf, _Tp &__Sf)`

This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.

- `template<typename _Tp >`
`void __fresnel_series (const _Tp __ax, _Tp &__Cf, _Tp &__Sf)`

This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.

- `template<typename _Tp >`
`_Tp __gamma (_Tp __a)`

Return the gamma function $\Gamma(a)$. The gamma function is defined by:

$$\Gamma(a) = \int_0^{\infty} e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __gamma (_Tp __a, _Tp __x)`

Return the incomplete gamma functions.

- `template<typename _Tp >`
`_Tp __gamma_cdf (_Tp __alpha, _Tp __beta, _Tp __x)`

Return the gamma cumulative propability distribution function.

- `template<typename _Tp >`
`_Tp __gamma_cdfc (_Tp __alpha, _Tp __beta, _Tp __x)`

Return the gamma complementary cumulative propability distribution function.

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __gamma_cont_frac (_Tp __a, _Tp __x)`

Return the incomplete gamma function by continued fraction.

- `template<typename _Tp >`
`_Tp __gamma_pdf (_Tp __alpha, _Tp __beta, _Tp __x)`

Return the gamma propability distribution function.

- `template<typename _Tp >`
`_Tp __gamma_reciprocal (_Tp __a)`
- `template<typename _Tp >`
`_Tp __gamma_reciprocal_series (_Tp __a)`

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __gamma_series (_Tp __a, _Tp __x)`

Return the incomplete gamma function by series summation.

$$\gamma(a, x) = x^a e^{-x} \sum_{k=1}^{\infty} \frac{x^k}{(a)_k}$$

- `template<typename _Tp >`
`__gnu_cxx::__gamma_temme_t< _Tp > __gamma_temme (_Tp __mu)`

Compute the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ 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 __gauss (_Tp __x)`
- `template<typename _Tp >`
`_Tp __gegenbauer_poly (unsigned int __n, _Tp __alpha, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel
(std::complex< _Tp > __nu, std::complex< _Tp > __z)`

- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel←
__debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char
& __aorb, int & __morn)`

- `template<typename _Tp >`
`void __hankel_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > & __p,
std::complex< _Tp > & __p2, std::complex< _Tp > & __nup2, std::complex< _Tp > & __num2, std::complex<
_Tp > & __num1d3, std::complex< _Tp > & __num2d3, std::complex< _Tp > & __num4d3, std::complex< ←
_Tp > & __zeta, std::complex< _Tp > & __zetaphf, std::complex< _Tp > & __zetamhf, std::complex< _Tp >
& __zetam3hf, std::complex< _Tp > & __zetrat)`

Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.

- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel←
__uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z)`

This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.

- template<typename _Tp >
[__gnu_cxx::__cyl_hankel_t](#)< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > [__hankel_uniform_olver](#) (std::complex< _Tp > __nu, std::complex< _Tp > __z)
 Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order *nu* along with their derivatives.
- template<typename _Tp >
 void [__hankel_uniform_outer](#) (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__etm3h, std::complex< _Tp > &__etrat, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)
 Compute outer factors and associated functions of *z* and *nu* appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of *z* and *nu* returned by [hankel_uniform_outer](#) are available for use in computing further terms in the expansions.
- template<typename _Tp >
 void [__hankel_uniform_sum](#) (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > &__num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > __Aip, std::complex< _Tp > __o4dp, std::complex< _Tp > __Aim, std::complex< _Tp > __o4dm, std::complex< _Tp > __od2p, std::complex< _Tp > __od0dp, std::complex< _Tp > __od2m, std::complex< _Tp > __od0dm, _Tp __eps, std::complex< _Tp > &__H1sum, std::complex< _Tp > &__H1psum, std::complex< _Tp > &__H2sum, std::complex< _Tp > &__H2psum)
 Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to *nterms* (less than 5) to achieve relative error *eps*.
- template<typename _Tp >
 _Tp [__harmonic_number](#) (unsigned int __n)
- template<typename _Tp >
 std::vector< [__gnu_cxx::__quadrature_point_t](#)< _Tp > > [__hermite_zeros](#) (unsigned int __n, _Tp __proto=[__Tp{}](#))
- template<typename _Tp >
 _Tp [__heuman_lambda](#) (_Tp __k, _Tp __phi)
- template<typename _Tp >
 _Tp [__hurwitz_zeta](#) (_Tp __s, _Tp __a)
 Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.
- template<typename _Tp >
 _Tp [__hurwitz_zeta_euler_maclaurin](#) (_Tp __s, _Tp __a)
 Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.
- template<typename _Tp >
 std::complex< _Tp > [__hurwitz_zeta_polylog](#) (_Tp __s, std::complex< _Tp > __a)
- template<typename _Tp >
 std::complex< _Tp > [__hydrogen](#) (unsigned int __n, unsigned int __l, unsigned int __m, _Tp __Z, _Tp __r, _Tp __theta, _Tp __phi)
- template<typename _Tp >
 _Tp [__hyperg](#) (_Tp __a, _Tp __b, _Tp __c, _Tp __x)
 Return the hypergeometric function ${}_2F_1(a, b; c; x)$.
- template<typename _Tp >
 _Tp [__hyperg_luke](#) (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)
 Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by an iterative procedure described in Luke, *Algorithms for the Computation of Mathematical Functions*.
- template<typename _Tp >
 _Tp [__hyperg_reflect](#) (_Tp __a, _Tp __b, _Tp __c, _Tp __x)
 Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by the reflection formulae in Abramowitz & Stegun formula 15.3.6 for $d = c - a - b$ not integral and formula 15.3.11 for $d = c - a - b$ integral. This assumes $a, b, c \neq$ negative integer.

- `template<typename _Tp >`
`_Tp __hyperg_series (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`
Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by series expansion.
- `template<typename _Tp >`
`_Tp __ibeta_cont_frac (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__jacobi_t<_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 >`
`std::vector< __gnu_cxx::__quadrature_point_t<_Tp > > __laguerre_zeros (unsigned int __n, _Tp __alpha)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __lanczos_binet1p (_Tp __z)`
Return the Binet function $J(1+z)$ by the Lanczos method. The Binet function is the log of the scaled Gamma function $\log(\Gamma^(z))$ defined by*

$$J(z) = \log(\Gamma^*(z)) = \log(\Gamma(z)) + z - \left(z - \frac{1}{2}\right) \log(z) - \log(2\pi)$$

or

$$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

where $\Gamma(z)$ is the gamma function.

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __lanczos_log_gamma1p (_Tp __z)`
Return the logarithm of the gamma function $\log(\Gamma(1+z))$ by the Lanczos method.
- `template<typename _Tp >`
`_Tp __legendre_q (unsigned int __l, _Tp __x)`
Return the Legendre function of the second kind by upward recursion on order l .
- `template<typename _Tp >`
`std::vector< __gnu_cxx::__quadrature_point_t<_Tp > > __legendre_zeros (unsigned int __l, _Tp proto=_Tp{})`
- `template<typename _Tp >`
`_Tp __log_binomial (unsigned int __n, unsigned int __k)`
Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

- `template<typename _Tp >`
`_Tp __log_binomial (_Tp __nu, unsigned int __k)`
Return the logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

- `template<typename _Tp >`
`_Tp __log_binomial_sign (_Tp __nu, unsigned int __k)`

Return the sign of the exponentiated logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu + 1)}{\Gamma(\nu - k + 1)\Gamma(k + 1)}$$

The binomial coefficients are generated by:

$$(1 + t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

- `template<typename _Tp >`
`std::complex< _Tp > __log_binomial_sign (std::complex< _Tp > __nu, unsigned int __k)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (_Tp __x)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (int __n)`

Return the logarithm of the double factorial of the integer n.

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_factorial (unsigned int __n)`

Return the logarithm of the factorial of the integer n.

- `template<typename _Tp >`
`_Tp __log_falling_factorial (_Tp __a, _Tp __nu)`

Return the logarithm of the falling factorial function or the lower Pochhammer symbol. The lower Pochhammer symbol is defined by

$$a^{\underline{n}} = \Gamma(a + 1) / \Gamma(a - \nu + 1) = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1$$

In particular, $n^{\underline{n}} = n!$. Thus this function returns

$$\ln[a^{\underline{n}}] = \ln[\Gamma(a + 1)] - \ln[\Gamma(a - \nu + 1)], \ln[a^{\underline{0}}] = 0$$

Many notations exist for this function:

$$(a)_\nu$$

,

$$\left\{ \begin{matrix} a \\ \nu \end{matrix} \right\}$$

, and others.

- `template<typename _Tp >`
`_Tp __log_gamma (_Tp __a)`

Return $\log(|\Gamma(a)|)$. This will return values even for $a < 0$. To recover the sign of $\Gamma(a)$ for any argument use `__log_gamma_sign`.

- `template<typename _Tp >`
`std::complex< _Tp > __log_gamma (std::complex< _Tp > __a)`

Return $\log(\Gamma(a))$ for complex argument.

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_bernoulli (_Tp __x)`

Return $\log(\Gamma(x))$ by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.

- `template<typename _Tp >`
`_Tp __log_gamma_sign (_Tp __a)`

Return the sign of $\Gamma(x)$. At nonpositive integers zero is returned indicating $\Gamma(x)$ is undefined.

- `template<typename _Tp >`
`std::complex< _Tp > __log_gamma_sign (std::complex< _Tp > __a)`

- `template<typename _Tp >`
`_Tp __log_rising_factorial (_Tp __a, _Tp __nu)`

Return the logarithm of the rising factorial function or the (upper) Pochhammer symbol. The Pochhammer symbol is defined for integer order by

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(a) = \prod_{k=0}^{\nu-1} (a + k), (a)_0 = 1$$

Thus this function returns

$$\ln[a^{\overline{\nu}}] = \ln[\Gamma(a + \nu)] - \ln[\Gamma(a)], \ln[(a)_0] = 0$$

Many notations exist for this function:

$$(a)_{\nu}$$

(especially in the literature of special functions),

$$\left[\begin{matrix} a \\ \nu \end{matrix} \right]$$

, and others.

- `template<typename _Tp >`
`_Tp __log_stirling_1 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __log_stirling_1_sign (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __log_stirling_2 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __logint (const _Tp __x)`

Return the logarithmic integral $li(x)$.

- `template<typename _Tp >`
`_Tp __logistic_cdf (_Tp __a, _Tp __b, _Tp __x)`

Return the logistic cumulative distribution function.

- `template<typename _Tp >`
`_Tp __logistic_pdf (_Tp __a, _Tp __b, _Tp __x)`

Return the logistic probability density function.

- `template<typename _Tp >`
`_Tp __lognormal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)`

Return the lognormal cumulative probability density function.

- `template<typename _Tp >`
`_Tp __lognormal_pdf (_Tp __mu, _Tp __sigma, _Tp __x)`

Return the lognormal probability density function.

- `template<typename _Tp >`
`_Tp __normal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)`

Return the normal cumulative probability density function.

- `template<typename _Tp >`
`_Tp __normal_pdf (_Tp __mu, _Tp __sigma, _Tp __x)`

Return the normal probability density function.

- `template<typename _Tp >`
`_Tp __owens_t (_Tp __h, _Tp __a)`

- `template<typename _Tp >`
`_Tp __pgamma (_Tp __a, _Tp __x)`

Return the regularized lower incomplete gamma function. The regularized lower incomplete gamma function is defined by

$$P(a, x) = \frac{\gamma(a, x)}{\Gamma(a)}$$

where $\Gamma(a)$ is the gamma function and

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

is the lower incomplete gamma function.

- `template<typename _Tp >`
`std::complex< _Tp > __polar_pi (_Tp __rho, _Tp __phi_pi)`
- `template<typename _Tp >`
`_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_prob_hermite_recursion (unsigned int __n, _Tp __x)`
This routine returns the Probabilists Hermite polynomial of order n: $He_n(x)$ by recursion on n.
- `template<typename _Tp >`
`_Tp __poly_radial_jacobi (unsigned int __n, unsigned int __m, _Tp __rho)`
- `template<typename _Tp >`
`_Tp __polylog (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp, typename _ArgType >`
`__gnu_cxx::__promote_fp_t< std::complex< _Tp >, _ArgType > __polylog_exp (_Tp __s, _ArgType __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg (int __n, std::complex< _Tp > __w)`

- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg_int` (int __s, std::complex< _Tp > __w)
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg_int` (int __s, _Tp __w)
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg_real` (_Tp __s, std::complex< _Tp > __w)
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg_real` (_Tp __s, _Tp __w)
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos` (unsigned int __s, std::complex< _Tp > __w)
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos` (unsigned int __s, _Tp __w)
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos` (_Tp __s, std::complex< _Tp > __w)
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos_int` (unsigned int __s, std::complex< _Tp > __w)
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos_int` (unsigned int __s, _Tp __w)
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos_real` (_Tp __s, std::complex< _Tp > __w)
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos_real` (_Tp __s, _Tp __w)
- `template<typename _PowTp, typename _Tp >`
`_Tp __polylog_exp_sum` (_PowTp __s, _Tp __w)
- `template<typename _Tp >`
`_Tp __psi` (unsigned int __n)

Return the digamma function of integral argument. The digamma or $\psi(x)$ function is defined as the logarithmic derivative of the gamma function:

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

The digamma series for integral argument is given by:

$$\psi(n) = -\gamma_E + \sum_{k=1}^{n-1} \frac{1}{k}$$

The latter sum is called the harmonic number, H_n .

- `template<typename _Tp >`
`_Tp __psi` (_Tp __x)

Return the digamma function. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

For negative argument the reflection formula is used:

$$\psi(x) = \psi(1-x) - \pi \cot(\pi x)$$

- `template<typename _Tp >`
`_Tp __psi` (unsigned int __n, _Tp __x)
Return the polygamma function $\psi^{(n)}(x)$.
- `template<typename _Tp >`
`_Tp __psi_asymp` (_Tp __x)

Return the digamma function for large argument. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

- `template<typename _Tp >`
`_Tp __psi_series (_Tp __x)`

Return the digamma function by series expansion. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

- `template<typename _Tp >`
`_Tp __qgamma (_Tp __a, _Tp __x)`

Return the regularized upper incomplete gamma function. The regularized upper incomplete gamma function is defined by

$$Q(a, x) = \frac{\Gamma(a, x)}{\Gamma(a)}$$

where $\Gamma(a)$ is the gamma function and

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1} dt (a > 0)$$

is the upper incomplete gamma function.

- `template<typename _Tp >`
`_Tp __rice_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`

Return the Rice probability density function.

- `template<typename _Tp >`
`_Tp __riemann_zeta (_Tp __s)`

Return the Riemann zeta function $\zeta(s)$.

- `template<typename _Tp >`
`_Tp __riemann_zeta_euler_maclaurin (_Tp __s)`

Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.

- `template<typename _Tp >`
`_Tp __riemann_zeta_glob (_Tp __s)`

- `template<typename _Tp >`
`_Tp __riemann_zeta_m_1 (_Tp __s)`

Return the Riemann zeta function $\zeta(s) - 1$.

- `template<typename _Tp >`
`_Tp __riemann_zeta_m_1_glob (_Tp __s)`

Evaluate the Riemann zeta function by series for all $s \neq 1$. Convergence is great until largish negative numbers. Then the convergence of the > 0 sum gets better.

- `template<typename _Tp >`
`_Tp __riemann_zeta_product (_Tp __s)`

Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.

- `template<typename _Tp >`
`_Tp __riemann_zeta_sum (_Tp __s)`

Compute the Riemann zeta function $\zeta(s)$ by summation for $s > 1$.

- `template<typename _Tp >`
`_Tp __rising_factorial (_Tp __a, int __n)`

Return the (upper) Pochhammer function or the rising factorial function. The Pochhammer symbol is defined by

$$a^{\overline{n}} = \Gamma(a + \nu) / \Gamma(\nu) = \prod_{k=0}^{n-1} (a + k), (a)_0 = 1$$

Many notations exist for this function:

$$(a)_\nu$$

, (especially in the literature of special functions),

$$\left[\begin{array}{c} a \\ n \end{array} \right]$$

, and others.

- `template<typename _Tp >`
`_Tp __rising_factorial (_Tp __a, _Tp __nu)`

Return the rising factorial function or the (upper) Pochhammer function. The rising factorial function is defined by

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(\nu)$$

Many notations exist for this function:

$$(a)_\nu$$

, (especially in the literature of special functions),

$$\left[\begin{array}{c} a \\ n \end{array} \right]$$

, and others.

- `template<typename _Tp >`
`_Tp __sin_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __sin_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __sinc (_Tp __x)`

Return the sinus cardinal function

$$\text{sinc}(x) = \frac{\sin(x)}{x}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __sinc_pi (_Tp __x)`

Return the reperiodized sinus cardinal function

$$\text{sinc}_\pi(x) = \frac{\sin(\pi x)}{\pi x}$$

- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< _Tp > __sincos (_Tp __x)`
- `template<>`
`__gnu_cxx::__sincos_t< float > __sincos (float __x)`
- `template<>`
`__gnu_cxx::__sincos_t< double > __sincos (double __x)`
- `template<>`
`__gnu_cxx::__sincos_t< long double > __sincos (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< _Tp > __sincos_pi (_Tp __x)`
- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __sincosint (_Tp __x)`

This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a pair.

- `template<typename _Tp >`
`void __sincosint_asymp (_Tp __t, _Tp &_Si, _Tp &_Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.

- `template<typename _Tp >`
`void __sincosint_cont_frac (_Tp __t, _Tp &_Si, _Tp &_Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.

- `template<typename _Tp >`
`void __sincosint_series (_Tp __t, _Tp &_Si, _Tp &_Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.

- `template<typename _Tp >`
`_Tp __sinh_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __sinh_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __sinhc (_Tp __x)`

Return the hyperbolic sinus cardinal function

$$\operatorname{sinhc}(x) = \frac{\sinh(x)}{x}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __sinhc_pi (_Tp __x)`

Return the reperiodized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_\pi(x) = \frac{\sinh(\pi x)}{\pi x}$$

- `template<typename _Tp >`
`_Tp __sinhint (const _Tp __x)`

Return the hyperbolic sine integral $Shi(x)$.

- `template<typename _Tp >`
`_Tp __sph_bessel (unsigned int __n, _Tp __x)`

Return the spherical Bessel function $j_n(x)$ of order n and non-negative real argument x .

- `template<typename _Tp >`
`std::complex< _Tp > __sph_bessel (unsigned int __n, std::complex< _Tp > __z)`

Return the complex spherical Bessel function.

- `template<typename _Tp >`
`__gnu_cxx::__sph_mod_bessel_t< unsigned int, _Tp, _Tp > __sph_bessel_ik (unsigned int __n, _Tp __x)`

Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i'_n(x)$ and $k'_n(x)$ respectively.

- `template<typename _Tp >`
`__gnu_cxx::__sph_bessel_t< unsigned int, _Tp, _Tp > __sph_bessel_jn (unsigned int __n, _Tp __x)`

Compute the spherical Bessel $j_n(x)$ and Neumann $n_n(x)$ functions and their first derivatives $j'_n(x)$ and $n'_n(x)$ respectively.

- `template<typename _Tp >`
`__gnu_cxx::__sph_bessel_t< unsigned int, _Tp, std::complex< _Tp > > __sph_bessel_jn_neg_arg (unsigned int __n, _Tp __x)`

- `template<typename _Tp >`
`__gnu_cxx::__sph_hankel_t< unsigned int, std::complex< _Tp >, std::complex< _Tp > > __sph_hankel (unsigned int __n, std::complex< _Tp > __z)`

Helper to compute complex spherical Hankel functions and their derivatives.

- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_1 (unsigned int __n, _Tp __x)`

Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.

- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_1 (unsigned int __n, std::complex< _Tp > __z)`

Return the complex spherical Hankel function of the first kind.

- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_2 (unsigned int __n, _Tp __x)`

Return the spherical Hankel function of the second kind $h_n^{(2)}(x)$.

- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)`

Return the complex spherical Hankel function of the second kind.

- `template<typename _Tp >`
`std::complex< _Tp > __sph_harmonic (unsigned int __l, int __m, _Tp __theta, _Tp __phi)`

Return the spherical harmonic function.

- `template<typename _Tp >`
`_Tp __sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`

Return the spherical associated Legendre function.

- `template<typename _Tp >`
`_Tp __sph_neumann (unsigned int __n, _Tp __x)`

Return the spherical Neumann function $n_n(x)$ of order n and non-negative real argument x .

- `template<typename _Tp >`
`std::complex< _Tp > __sph_neumann (unsigned int __n, std::complex< _Tp > __z)`

Return the complex spherical Neumann function.

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __spouge_binet1p (_Tp __z)`

Return the Binet function $J(1+z)$ by the Spouge method. The Binet function is the log of the scaled Gamma function $\log(\Gamma^(z))$ defined by*

$$J(z) = \log(\Gamma^*(z)) = \log(\Gamma(z)) + z - \left(z - \frac{1}{2}\right) \log(z) - \log(2\pi)$$

or

$$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

where $\Gamma(z)$ is the gamma function.

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __spouge_log_gamma1p (_Tp __z)`

Return the logarithm of the gamma function $\log(\Gamma(1+z))$ by the Spouge algorithm:

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[\sqrt{2\pi} + \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

- `template<typename _Tp >`
`_Tp __stirling_1 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __stirling_1_recur (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __stirling_1_series (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __stirling_2 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __stirling_2_recur (unsigned int __n, unsigned int __m)`

- `template<typename _Tp >`
`_Tp __stirling_2_series` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp __student_t_cdf` (_Tp __t, unsigned int __nu)
Return the Students T probability function.
- `template<typename _Tp >`
`_Tp __student_t_cdfc` (_Tp __t, unsigned int __nu)
Return the complement of the Students T probability function.
- `template<typename _Tp >`
`_Tp __student_t_pdf` (_Tp __t, unsigned int __nu)
Return the Students T probability density.
- `template<typename _Tp >`
`_Tp __tan_pi` (_Tp __x)
- `template<typename _Tp >`
`std::complex< _Tp > __tan_pi` (std::complex< _Tp > __z)
- `template<typename _Tp >`
`_Tp __tanh_pi` (_Tp __x)
- `template<typename _Tp >`
`std::complex< _Tp > __tanh_pi` (std::complex< _Tp > __z)
- `template<typename _Tp >`
`_Tp __tgamma` (_Tp __a, _Tp __x)
Return the upper incomplete gamma function. The lower incomplete gamma function is defined by

$$\Gamma(a, x) = \int_x^{\infty} e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`
`_Tp __tgamma_lower` (_Tp __a, _Tp __x)
Return the lower incomplete gamma function. The lower incomplete gamma function is defined by

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`
`_Tp __theta_1` (_Tp __nu, _Tp __x)
- `template<typename _Tp >`
`_Tp __theta_2` (_Tp __nu, _Tp __x)
- `template<typename _Tp >`
`_Tp __theta_2_asymp` (_Tp __nu, _Tp __x)
- `template<typename _Tp >`
`_Tp __theta_2_sum` (_Tp __nu, _Tp __x)
- `template<typename _Tp >`
`_Tp __theta_3` (_Tp __nu, _Tp __x)
- `template<typename _Tp >`
`_Tp __theta_3_asymp` (_Tp __nu, _Tp __x)
- `template<typename _Tp >`
`_Tp __theta_3_sum` (_Tp __nu, _Tp __x)
- `template<typename _Tp >`
`_Tp __theta_4` (_Tp __nu, _Tp __x)
- `template<typename _Tp >`
`_Tp __theta_c` (_Tp __k, _Tp __x)
- `template<typename _Tp >`
`_Tp __theta_d` (_Tp __k, _Tp __x)

- `template<typename _Tp >`
`_Tp __theta_n (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_s (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __tricomi_u (_Tp __a, _Tp __c, _Tp __x)`

Return the Tricomi confluent hypergeometric function

$$U(a, c, x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a; c; x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1; 2-c; x)$$

- `template<typename _Tp >`
`_Tp __tricomi_u_naive (_Tp __a, _Tp __c, _Tp __x)`

Return the Tricomi confluent hypergeometric function

$$U(a, c, x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a; c; x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1; 2-c; x)$$

- `template<typename _Tp >`
`_Tp __weibull_cdf (_Tp __a, _Tp __b, _Tp __x)`
Return the Weibull cumulative probability density function.
- `template<typename _Tp >`
`_Tp __weibull_pdf (_Tp __a, _Tp __b, _Tp __x)`
Return the Weibull probability density function.
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t<_Tp> __zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)`
- `template<typename _Tp >`
`_Tp __znorm1 (_Tp __x)`
- `template<typename _Tp >`
`_Tp __znorm2 (_Tp __x)`

Variables

- `template<typename _Tp >`
`constexpr int __max_FGH = _Airy_series<_Tp>::__N_FGH`
- `template<>`
`constexpr int __max_FGH<double> = 79`
- `template<>`
`constexpr int __max_FGH<float> = 15`
- `constexpr size_t _Num_Euler_Maclaurin_zeta = 100`
- `constexpr _Factorial_table<long double> _S_double_factorial_table [301]`
- `constexpr long double _S_Euler_Maclaurin_zeta [_Num_Euler_Maclaurin_zeta]`
- `constexpr _Factorial_table<long double> _S_factorial_table [171]`
- `constexpr unsigned long long _S_harmonic_denom [_S_num_harmonic_numer]`
- `constexpr unsigned long long _S_harmonic_numer [_S_num_harmonic_numer]`
- `constexpr _Factorial_table<long double> _S_neg_double_factorial_table [999]`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_double_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_double_factorials<double> = 301`
- `template<>`
`constexpr std::size_t _S_num_double_factorials<float> = 57`

- `template<>`
`constexpr std::size_t _S_num_double_factorials< long double > = 301`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_factorials< double > = 171`
- `template<>`
`constexpr std::size_t _S_num_factorials< float > = 35`
- `template<>`
`constexpr std::size_t _S_num_factorials< long double > = 171`
- `constexpr unsigned long long _S_num_harmonic_numer = 29`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_neg_double_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< double > = 150`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< float > = 27`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< long double > = 999`
- `constexpr size_t _S_num_zetam1 = 121`
- `constexpr long double _S_zetam1 [_S_num_zetam1]`

9.3.1 Function Documentation

9.3.1.1 `template<typename _Tp > __gnu_cxx::__airy_t<_Tp, _Tp> std::__detail::__airy (_Tp __z)`

Compute the Airy functions $Ai(x)$ and $Bi(x)$ and their first derivatives $Ai'(x)$ and $Bi'(x)$ respectively.

Parameters

<code>__z</code>	The argument of the Airy functions.
------------------	-------------------------------------

Returns

A struct containing the Airy functions of the first and second kinds and their derivatives.

Definition at line 466 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`, and `__cyl_bessel_jn()`.

Referenced by `__airy_ai()`, `__airy_bi()`, `__fock_airy()`, and `__poly_hermite_asymp()`.

9.3.1.2 `template<typename _Tp > std::complex<_Tp> std::__detail::__airy_ai (std::complex<_Tp > __z)`

Return the complex Airy Ai function.

Definition at line 2622 of file `sf_airy.tcc`.

References `__airy()`.

9.3.1.3 `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 215 of file `sf_hankel.tcc`.

Referenced by `__hankel_uniform_outer()`.

9.3.1.4 `template<typename _Tp> std::complex<_Tp> std::__detail::__airy_bi (std::complex<_Tp> __z)`

Return the complex Airy Bi function.

Definition at line 2634 of file `sf_airy.tcc`.

References `__airy()`.

9.3.1.5 `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

\leftrightarrow _n	The order
\leftrightarrow _m	The degree
\leftrightarrow _x	The argument

Returns

The value of the associated Laguerre polynomial of order n, degree m, and argument x.

Definition at line 364 of file sf_laguerre.tcc.

Referenced by __hydrogen().

9.3.1.6 `template<typename _Tp > _Tp std::__detail::__assoc_legendre_p (unsigned int __l, unsigned int __m, _Tp __x)`

Return the associated Legendre function by recursion on l and downward recursion on m .

The associated Legendre function is derived from the Legendre function $P_l(x)$ by the Rodrigues formula:

$$P_l^m(x) = (1 - x^2)^{m/2} \frac{d^m}{dx^m} P_l(x)$$

Parameters

\leftrightarrow _l	The order of the associated Legendre function. $l \geq 0$.
\leftrightarrow _m	The order of the associated Legendre function. $m \leq l$.
\leftrightarrow _x	The argument of the associated Legendre function.

Definition at line 183 of file sf_legendre.tcc.

References __poly_legendre_p().

9.3.1.7 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (unsigned int __n)`

This returns Bernoulli number B_n .

Parameters

\leftrightarrow	the order n of the Bernoulli number.
n	

Returns

The Bernoulli number of order n.

Definition at line 128 of file sf_bernoulli.tcc.

Referenced by `__euler()`, and `__gnu_cxx::bernoulli()`.

9.3.1.8 `template<typename _Tp> _Tp std::__detail::__bernoulli (unsigned int __n, _Tp __x)`

Return the Bernoulli polynomial $B_n(x)$ of order n at argument x.

The values at 0 and 1 are equal to the corresponding Bernoulli number:

$$B_n(0) = B_n(1) = B_n$$

The derivative is proportional to the previous polynomial:

$$B'_n(x) = n * B_{n-1}(x)$$

The series expansion is:

$$B_n(x) = \sum_{k=0}^n B_k \text{binom} n k x^{n-k}$$

A useful argument promotion is:

$$B_n(x+1) - B_n(x) = n * x^{n-1}$$

Definition at line 168 of file sf_bernoulli.tcc.

References `__binomial()`.

9.3.1.9 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (unsigned int __n)`

This returns Bernoulli number B_{2n} at even integer arguments $2n$.

Parameters

\leftrightarrow	the half-order n of the Bernoulli number.
n	

Returns

The Bernoulli number of order $2n$.

Definition at line 140 of file sf_bernoulli.tcc.

9.3.1.10 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series (unsigned int __n)`

This returns Bernoulli numbers from a table or by summation for larger values.

$$B_{2n} = (-1)^{n+1} 2 \frac{(2n)!}{(2\pi)^{2n}} \zeta(2n)$$

Note that

$$\zeta(2n) - 1 = (-1)^{n+1} \frac{(2\pi)^{2n}}{(2n)!} B_{2n} - 2$$

are small and rapidly decreasing functions of n .

Parameters

<code>__n</code>	the order n of the Bernoulli number.
------------------	--

Returns

The Bernoulli number of order n .

Definition at line 65 of file sf_bernoulli.tcc.

9.3.1.11 `template<typename _Tp > _Tp std::__detail::__beta (_Tp __a, _Tp __b)`

Return the beta function $B(a, b)$.

The beta function is defined by

$$B(a, b) = \int_0^1 t^{a-1} (1-t)^{b-1} dt = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Parameters

<code>__a</code>	The first argument of the beta function.
<code>__b</code>	The second argument of the beta function.

Returns

The beta function.

Definition at line 215 of file sf_beta.tcc.

References `__beta_gamma()`, and `__beta_lgamma()`.

Referenced by `__fisher_f_pdf()`, `__poly_jacobi()`, `__gnu_cxx::gamma_cdf()`, `__gnu_cxx::gamma_pdf()`, `__gnu_cxx::jacobi()`, `__gnu_cxx::jacobif()`, `__gnu_cxx::jacobil()`, and `std::__detail::_Airy<_Tp>::operator()`.

9.3.1.12 template<typename _Tp> _Tp std::__detail::__beta_gamma (_Tp __a, _Tp __b)

Return the beta function: $B(a, b)$.

The beta function is defined by

$$B(a, b) = \int_0^1 t^{a-1} (1-t)^{b-1} dt = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Parameters

<code>__a</code>	The first argument of the beta function.
<code>__b</code>	The second argument of the beta function.

Returns

The beta function.

Definition at line 77 of file sf_beta.tcc.

References `__gamma()`.

Referenced by `__beta()`.

9.3.1.13 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 311 of file sf_beta.tcc.

References __ibeta_cont_frac(), __log_gamma(), and __log_gamma_sign().

Referenced by __binomial_cdf(), __binomial_cdfc(), __fisher_f_cdf(), __fisher_f_cdfc(), __student_t_cdf(), and \leftrightarrow student_t_cdfc().

9.3.1.14 `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 125 of file sf_beta.tcc.

References __log_gamma(), and __log_gamma_sign().

Referenced by __beta().

9.3.1.15 `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:

$$B(a, b) = \frac{a+b}{ab} \prod_{k=1}^{\infty} \frac{1 + (a+b)/k}{(1 + a/k)(1 + b/k)} = \frac{a+b}{ab} \prod_{k=1}^{\infty} \left[1 - \frac{ab}{(a+k)(b+k)} \right]$$

Parameters

\leftarrow _a	The first argument of the beta function.
\leftarrow _b	The second argument of the beta function.

Returns

The beta function.

Definition at line 179 of file sf_beta.tcc.

9.3.1.16 `template<typename _Tp> _Tp std::__detail::__binomial (unsigned int __n, unsigned int __k)`

Return the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

Parameters

\leftarrow _n	The first argument of the binomial coefficient.
\leftarrow _k	The second argument of the binomial coefficient.

Returns

The binomial coefficient.

Definition at line 2515 of file sf_gamma.tcc.

References `std::__detail::__Factorial_table<_Tp>::__n`.

Referenced by `__bernoulli()`.

9.3.1.17 `template<typename _Tp> _Tp std::__detail::__binomial (_Tp __nu, unsigned int __k)`

Return the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu + 1)}{\Gamma(\nu - k + 1)\Gamma(k + 1)}$$

The binomial coefficients are generated by:

$$(1 + t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

Parameters

<code>__nu</code>	The real first argument of the binomial coefficient.
<code>__k</code>	The second argument of the binomial coefficient.

Returns

The binomial coefficient.

Definition at line 2560 of file `sf_gamma.tcc`.

References `__gamma()`, `__log_binomial()`, `__log_binomial_sign()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

9.3.1.18 `template<typename _Tp> _Tp std::__detail::__binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)`

Return the binomial cumulative distribution function.

The binomial cumulative distribution function is related to the incomplete beta function:

$$P(k|n, p) = I_p(k, n - k + 1)$$

Parameters

<code>__p</code>	
<code>__n</code>	
<code>__k</code>	

Definition at line 614 of file `sf_distributions.tcc`.

References `__beta_inc()`.

9.3.1.19 `template<typename _Tp> _Tp std::__detail::__binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)`

Return the complementary binomial cumulative distribution function.

The binomial cumulative distribution function is related to the incomplete beta function:

$$Q(k|n, p) = I_{1-p}(n - k + 1, k)$$

Parameters

\overleftrightarrow{p}	
\overleftrightarrow{n}	
\overleftrightarrow{k}	

Definition at line 644 of file sf_distributions.tcc.

References `__beta_inc()`.

9.3.1.20 `template<typename _Tp> _Tp std::__detail::__binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)`

Return the binomial probability mass function.

The binomial cumulative distribution function is related to the incomplete beta function:

$$f(k|n, p) = \binom{n}{k} p^k (1 - p)^{n-k}$$

Parameters

\overleftrightarrow{p}	
\overleftrightarrow{n}	
\overleftrightarrow{k}	

Definition at line 578 of file sf_distributions.tcc.

9.3.1.21 `template<typename _Sp, typename _Tp> _Tp std::__detail::__bose_einstein (_Sp __s, _Tp __x)`

Return the Bose-Einstein integral of integer or real order s and real argument x.

See also

https://en.wikipedia.org/wiki/Clausen_function
<http://dlmf.nist.gov/25.12.16>

$$G_s(x) = \frac{1}{\Gamma(s+1)} \int_0^\infty \frac{t^s}{e^{t-x} - 1} dt = Li_{s+1}(e^x)$$

Parameters

\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 1424 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.1.22 `template<typename _Tp> _Tp std::__detail::__chebyshev_recur (unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1)`

Return a Chebyshev polynomial of non-negative order n and real argument x by the recursion

$$C_n(x) = 2xC_{n-1} - C_{n-2}$$

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

\leftrightarrow _n	The non-negative integral order
\leftrightarrow _x	The real argument $-1 \leq x \leq +1$
<code>__C0</code>	The value of the zeroth-order Chebyshev polynomial at x
<code>__C1</code>	The value of the first-order Chebyshev polynomial at x

Definition at line 59 of file sf_chebyshev.tcc.

Referenced by `__chebyshev_t()`, `__chebyshev_u()`, `__chebyshev_v()`, and `__chebyshev_w()`.

9.3.1.23 `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 87 of file `sf_chebyshev.tcc`.

References `__chebyshev_recur()`.

9.3.1.24 `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 116 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

9.3.1.25 `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 146 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

9.3.1.26 `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

\leftrightarrow _n	The non-negative integral order
\leftrightarrow _x	The real argument $-1 \leq x \leq +1$

Definition at line 176 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

9.3.1.27 `template<typename _Tp> _Tp std::__detail::__chi_squared_pdf (_Tp __chi2, unsigned int __nu)`

Return the chi-squared propability function. This returns the probability that the observed chi-squared for a correct model is less than the value χ^2 .

The chi-squared propability function is related to the normalized lower incomplete gamma function:

$$P(\chi^2|\nu) = \Gamma_P\left(\frac{\nu}{2}, \frac{\chi^2}{2}\right)$$

Definition at line 75 of file sf_distributions.tcc.

References `__pgamma()`.

9.3.1.28 `template<typename _Tp> _Tp std::__detail::__chi_squared_pdfc (_Tp __chi2, unsigned int __nu)`

Return the complementary chi-squared propability function. This returns the probability that the observed chi-squared for a correct model is greater than the value χ^2 .

The complementary chi-squared propability function is related to the normalized upper incomplete gamma function:

$$Q(\chi^2|\nu) = \Gamma_Q\left(\frac{\nu}{2}, \frac{\chi^2}{2}\right)$$

Definition at line 99 of file sf_distributions.tcc.

References `__qgamma()`.

9.3.1.29 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__chshint (_Tp __x, _Tp & __Chi, _Tp & __Shi)`

This function returns the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals as a pair.

The hyperbolic cosine integral is defined by:

$$Chi(x) = \gamma_E + \log(x) + \int_0^x dt \frac{\cosh(t) - 1}{t}$$

The hyperbolic sine integral is defined by:

$$Shi(x) = \int_0^x dt \frac{\sinh(t)}{t}$$

Definition at line 166 of file sf_hypint.tcc.

References `__chshint_cont_frac()`, and `__chshint_series()`.

9.3.1.30 `template<typename _Tp> void std::__detail::__chshint_cont_frac (_Tp __t, _Tp & _Chi, _Tp & _Shi)`

This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by continued fraction for positive argument.

Definition at line 53 of file sf_hypint.tcc.

Referenced by `__chshint()`.

9.3.1.31 `template<typename _Tp> void std::__detail::__chshint_series (_Tp __t, _Tp & _Chi, _Tp & _Shi)`

This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by series summation for positive argument.

Definition at line 96 of file sf_hypint.tcc.

Referenced by `__chshint()`.

9.3.1.32 `template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_0_m2pi (std::complex<_Tp> __z)`

Definition at line 147 of file sf_polylog.tcc.

Referenced by `__polylog_exp_neg_int()`, `__polylog_exp_neg_real()`, `__polylog_exp_pos_int()`, and `__polylog_exp_↔pos_real()`.

9.3.1.33 `template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_pi (std::complex<_Tp> __z)`

Definition at line 134 of file sf_polylog.tcc.

Referenced by `__polylog_exp_neg_int()`, `__polylog_exp_neg_real()`, `__polylog_exp_pos_int()`, and `__polylog_exp_↔pos_real()`.

9.3.1.34 `template<typename _Tp> std::complex<_Tp> std::__detail::__clausen (unsigned int __m, std::complex<_Tp> __z)`

Return Clausen's function of integer order m and complex argument z . The notation and connection to polylog is from Wikipedia

Parameters

<code>__↔ _m</code>	The non-negative integral order.
<code>__↔ _w</code>	The complex argument.

Returns

The complex Clausen function.

Definition at line 1219 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.1.35 `template<typename _Tp> _Tp std::__detail::__clausen (unsigned int __m, _Tp __x)`

Return Clausen's function of integer order m and real argument x . The notation and connection to polylog is from Wikipedia

Parameters

\leftarrow <code>__m</code>	The integer order $m \geq 1$.
\leftarrow <code>__x</code>	The real argument.

Returns

The Clausen function.

Definition at line 1246 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.1.36 `template<typename _Tp> _Tp std::__detail::__clausen_cl (unsigned int __m, std::complex<_Tp> __z)`

Return Clausen's cosine sum Cl_m for positive integer order m and complex argument w .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

\leftarrow <code>__m</code>	The integer order $m \geq 1$.
\leftarrow <code>__z</code>	The complex argument.

Returns

The Clausen cosine sum $Cl_m(w)$,

Definition at line 1330 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.1.37 `template<typename _Tp> _Tp std::__detail::__clausen_cl(unsigned int __m, _Tp __x)`

Return Clausen's cosine sum Cl_m for positive integer order m and real argument w .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

\leftrightarrow <code>__m</code>	The integer order $m \geq 1$.
\leftrightarrow <code>__x</code>	The real argument.

Returns

The real Clausen cosine sum $Cl_m(w)$,

Definition at line 1358 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.1.38 `template<typename _Tp> _Tp std::__detail::__clausen_sl(unsigned int __m, std::complex<_Tp> __z)`

Return Clausen's sine sum Sl_m for positive integer order m and complex argument z .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

\leftrightarrow <code>__m</code>	The integer order $m \geq 1$.
\leftrightarrow <code>__z</code>	The complex argument.

Returns

The Clausen sine sum $Sl_m(w)$,

Definition at line 1274 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.1.39 `template<typename _Tp> _Tp std::__detail::__clausen_sl (unsigned int __m, _Tp __x)`

Return Clausen's sine sum Sl_m for positive integer order m and real argument x .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

\longleftrightarrow __m	The integer order $m \geq 1$.
\longleftrightarrow __x	The real argument.

Returns

The Clausen sine sum $Sl_m(w)$,

Definition at line 1302 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.1.40 `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

\longleftrightarrow __k	The modulus of the complete elliptic function.
------------------------------	--

Returns

The complete elliptic function of the first kind.

Definition at line 568 of file sf_ellint.tcc.

References `__comp_ellint_rf()`.

Referenced by `__ellint_1()`, `__ellnome_k()`, `__heuman_lambda()`, `__jacobi_zeta()`, `__theta_c()`, `__theta_d()`, `__theta_↔n()`, and `__theta_s()`.

9.3.1.41 `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

<code>↔ __k</code>	The modulus of the complete elliptic function.
------------------------	--

Returns

The complete elliptic function of the second kind.

Definition at line 642 of file sf_ellint.tcc.

References `__ellint_rd()`, and `__ellint_rf()`.

Referenced by `__ellint_2()`.

9.3.1.42 `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 732 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

Referenced by `__ellint_3()`.

9.3.1.43 `template<typename _Tp> _Tp std::__detail::__comp_ellint_d (_Tp __k)`

Return the complete Legendre elliptic integral D.

Definition at line 840 of file `sf_ellint.tcc`.

References `__ellint_rd()`.

9.3.1.44 `template<typename _Tp> _Tp std::__detail::__comp_ellint_rf (_Tp __x, _Tp __y)`

Definition at line 238 of file `sf_ellint.tcc`.

Referenced by `__comp_ellint_1()`, and `__ellint_rf()`.

9.3.1.45 `template<typename _Tp> _Tp std::__detail::__comp_ellint_rg (_Tp __x, _Tp __y)`

Definition at line 349 of file `sf_ellint.tcc`.

Referenced by `__ellint_rg()`.

9.3.1.46 `template<typename _Tp> _Tp std::__detail::__conf_hyperg (_Tp __a, _Tp __c, _Tp __x)`

Return the confluent hypergeometric function ${}_1F_1(a; c; x) = M(a, c, x)$.

Parameters

<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()`.

Referenced by `__tricoli_u_naive()`.

9.3.1.47 `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()`.

9.3.1.48 `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().

9.3.1.49 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().

9.3.1.50 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

\longleftrightarrow __a	The "numerator" parameter.
\longleftrightarrow __c	The "denominator" parameter.
\longleftrightarrow __x	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 141 of file sf_hyperg.tcc.

Referenced by __conf_hyperg().

9.3.1.51 `template<typename _Tp> _Tp std::__detail::__cos_pi (_Tp __x)`

Return the reperiodized cosine of argument x:

$$\cos_{\pi}(x) = \cos(\pi x)$$

Definition at line 102 of file sf_trig.tcc.

Referenced by `__cos_pi()`, `__cosh_pi()`, `__cyl_bessel_jn()`, `__cyl_bessel_jn_neg_arg()`, `__log_double_factorial()`, `__sin_pi()`, and `__sinh_pi()`.

9.3.1.52 `template<typename _Tp> std::complex<_Tp> std::__detail::__cos_pi (std::complex<_Tp> __z)`

Return the reperiodized cosine of complex argument z:

$$\cos_{\pi}(z) = \cos(\pi z) = \cos_{\pi}(x)\cosh_{\pi}(y) - i\sin_{\pi}(x)\sinh_{\pi}(y)$$

Definition at line 227 of file sf_trig.tcc.

References `__cos_pi()`, and `__sin_pi()`.

9.3.1.53 `template<typename _Tp> _Tp std::__detail::__cosh_pi (_Tp __x)`

Return the reperiodized hyperbolic cosine of argument x:

$$\cosh_{\pi}(x) = \cosh(\pi x)$$

Definition at line 130 of file sf_trig.tcc.

9.3.1.54 `template<typename _Tp> std::complex<_Tp> std::__detail::__cosh_pi (std::complex<_Tp> __z)`

Return the reperiodized hyperbolic cosine of complex argument z:

$$\cosh_{\pi}(z) = \cosh_{\pi}(z) = \cosh_{\pi}(x)\cos_{\pi}(y) + i\sinh_{\pi}(x)\sin_{\pi}(y)$$

Definition at line 249 of file sf_trig.tcc.

References `__cos_pi()`, and `__sin_pi()`.

9.3.1.55 `template<typename _Tp> _Tp std::__detail::__coshint (const _Tp __x)`

Return the hyperbolic cosine integral $Chi(x)$.

The hyperbolic cosine integral is given by

$$Chi(x) = (Ei(x) - E_1(x))/2 = (Ei(x) + Ei(-x))/2$$

Parameters

<code>_↔</code>	The argument of the hyperbolic cosine integral function.
<code>_x</code>	

Returns

The hyperbolic cosine integral.

Definition at line 561 of file sf_expint.tcc.

References `__expint_E1()`, and `__expint_Ei()`.

9.3.1.56 `template<typename _Tp > std::complex<_Tp> std::__detail::__cyl_bessel (std::complex<_Tp > __nu, std::complex<_Tp > __z)`

Return the complex cylindrical Bessel function.

Parameters

in	<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 1174 of file sf_hankel.tcc.

References `__hankel()`.

9.3.1.57 `template<typename _Tp > _Tp std::__detail::__cyl_bessel_i (_Tp __nu, _Tp __x)`

Return the regular modified Bessel function of order ν : $I_\nu(x)$.

The regular modified cylindrical Bessel function is:

$$I_\nu(x) = \sum_{k=0}^{\infty} \frac{(x/2)^{\nu+2k}}{k! \Gamma(\nu + k + 1)}$$

Parameters

<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 364 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_ik()`.

Referenced by `__rice_pdf()`.

9.3.1.58 `template<typename _Tp> _Tp std::__detail::__cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)`

This routine returns the cylindrical Bessel functions of order ν : J_ν or I_ν by series expansion.

The modified cylindrical Bessel function is:

$$Z_\nu(x) = \sum_{k=0}^{\infty} \frac{\sigma^k (x/2)^{\nu+2k}}{k! \Gamma(\nu + k + 1)}$$

where $\sigma = +1$ or -1 for $Z = I$ or J respectively.

See Abramowitz & Stegun, 9.1.10 Abramowitz & Stegun, 9.6.7 (1) Handbook of Mathematical Functions, ed. Milton Abramowitz and Irene A. Stegun, Dover Publications, Equation 9.1.10 p. 360 and Equation 9.6.10 p. 375

Parameters

<code>__nu</code>	The order of the Bessel function.
<code>__x</code>	The argument of the Bessel function.
<code>__sgn</code>	The sign of the alternate terms -1 for the Bessel function of the first kind. +1 for the modified Bessel function of the first kind.
<code>__max_iter</code>	The maximum number of iterations for sum.

Returns

The output Bessel function.

Definition at line 413 of file sf_bessel.tcc.

References `__log_gamma()`.

Referenced by `__cyl_bessel_i()`, and `__cyl_bessel_j()`.

9.3.1.59 `template<typename _Tp> __gnu_cxx::__cyl_mod_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_ik (_Tp __nu, _Tp __x)`

Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.

Parameters

<code>__nu</code>	The order of the Bessel functions.
<code>__x</code>	The argument of the Bessel functions.

Returns

A struct containing the modified cylindrical Bessel functions of the first and second kinds and their derivatives.

Definition at line 302 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik_asymp()`, `__cyl_bessel_ik_steel()`, and `__sin_pi()`.

Referenced by `__airy()`, `__cyl_bessel_i()`, `__cyl_bessel_k()`, and `__sph_bessel_ik()`.

9.3.1.60 `template<typename _Tp> __gnu_cxx::__cyl_mod_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_ik_asymp (_Tp __nu, _Tp __x)`

This routine computes the asymptotic modified cylindrical Bessel and functions of order 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.

Returns

A struct containing the modified cylindrical Bessel functions of the first and second kinds and their derivatives.

Definition at line 79 of file `sf_mod_bessel.tcc`.

Referenced by `__cyl_bessel_ik()`, and `__cyl_bessel_ik_steel()`.

9.3.1.61 `template<typename _Tp> __gnu_cxx::__cyl_mod_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_ik_steel (_Tp __nu, _Tp __x)`

Compute the modified Bessel functions $I_\nu(x)$ and $K_\nu(x)$ and their first derivatives $I'_\nu(x)$ and $K'_\nu(x)$ respectively. These four functions are computed together for numerical stability.

Parameters

<code>__nu</code>	The order of the Bessel functions.
<code>__x</code>	The argument of the Bessel functions.

Returns

A struct containing the modified cylindrical Bessel functions of the first and second kinds and their derivatives.

Definition at line 145 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ik_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_ik()`.

9.3.1.62 `template<typename _Tp> _Tp std::__detail::__cyl_bessel_j (_Tp __nu, _Tp __x)`

Return the Bessel function of order ν : $J_\nu(x)$.

The cylindrical Bessel function is:

$$J_\nu(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{\nu+2k}}{k! \Gamma(\nu + k + 1)}$$

Parameters

<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 559 of file sf_bessel.tcc.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_jn()`.

9.3.1.63 `template<typename _Tp> __gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn (_Tp __nu, _Tp __x)`

Return the cylindrical Bessel functions and their derivatives of order ν by various means.

Definition at line 452 of file sf_bessel.tcc.

References `__cos_pi()`, `__cyl_bessel_jn_asymp()`, `__cyl_bessel_jn_steel()`, and `__sin_pi()`.

Referenced by `__airy()`, `__cyl_bessel_j()`, `__cyl_bessel_jn_neg_arg()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann_n()`, and `__sph_bessel_jn()`.

9.3.1.64 `template<typename _Tp> __gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn_asymp (_Tp __nu, _Tp __x)`

This routine computes the asymptotic cylindrical Bessel and Neumann functions of order 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.

Returns

A struct containing the cylindrical Bessel functions of the first and second kinds and their derivatives.

Definition at line 79 of file sf_bessel.tcc.

Referenced by `__cyl_bessel_jn()`, and `__cyl_bessel_jn_stepped()`.

```
9.3.1.65  template<typename _Tp> __gnu_cxx::__cyl_bessel_t<_Tp, _Tp, std::complex<_Tp>> >
          std::__detail::__cyl_bessel_jn_neg_arg ( _Tp __nu, _Tp __x )
```

Return the cylindrical Bessel functions and their derivatives of order ν and argument $x < 0$.

Definition at line 518 of file sf_bessel.tcc.

References `__cos_pi()`, `__cyl_bessel_jn()`, and `__polar_pi()`.

Referenced by `__cyl_hankel_1()`, `__cyl_hankel_2()`, and `__sph_bessel_jn_neg_arg()`.

```
9.3.1.66  template<typename _Tp> __gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn_stepped ( _Tp
          __nu, _Tp __x )
```

Compute the Bessel $J_\nu(x)$ and Neumann $N_\nu(x)$ functions and their first derivatives $J'_\nu(x)$ and $N'_\nu(x)$ respectively. These four functions are computed together for numerical stability.

Parameters

<code>__nu</code>	The order of the Bessel functions.
<code>__x</code>	The argument of the Bessel functions.

Returns

A struct containing the cylindrical Bessel functions of the first and second kinds and their derivatives.

Definition at line 199 of file sf_bessel.tcc.

References `__cyl_bessel_jn_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_jn()`.

9.3.1.67 `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 398 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`.

9.3.1.68 `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 616 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`, `__cyl_bessel_jn_neg_arg()`, and `__polar_pi()`.

9.3.1.69 `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 1140 of file `sf_hankel.tcc`.

References `__hankel()`.

9.3.1.70 `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 654 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`, `__cyl_bessel_jn_neg_arg()`, and `__polar_pi()`.

9.3.1.71 `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 1157 of file `sf_hankel.tcc`.

References `__hankel()`.

9.3.1.72 `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 1191 of file `sf_hankel.tcc`.

References `__hankel()`.

9.3.1.73 `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 590 of file sf_bessel.tcc.

References `__cyl_bessel_jn()`.

9.3.1.74 `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 235 of file sf_dawson.tcc.

References `__dawson_cont_frac()`, and `__dawson_series()`.

9.3.1.75 `template<typename _Tp> _Tp std::__detail::__dawson_cont_frac (_Tp __x)`

Compute the Dawson integral using a sampling theorem representation.

This array could be built on a thread-local basis.

Definition at line 73 of file sf_dawson.tcc.

Referenced by `__dawson()`.

9.3.1.76 `template<typename _Tp> _Tp std::__detail::__dawson_series (_Tp __x)`

Compute the Dawson integral using the series expansion.

Definition at line 49 of file sf_dawson.tcc.

Referenced by `__dawson()`.

9.3.1.77 `template<typename _Tp > _Tp std::__detail::__debye (unsigned int __n, _Tp __x)`

Return the Debye function. The Debye functions are related to the incomplete Riemann zeta function:

$$\zeta_x(s) = \frac{1}{\Gamma(s)} \int_0^x \frac{t^{s-1}}{e^t - 1} dt = \sum_{k=1}^{\infty} \frac{P(s, kx)}{k^s}$$

$$Z_x(s) = \frac{1}{\Gamma(s)} \int_x^{\infty} \frac{t^{s-1}}{e^t - 1} dt = \sum_{k=1}^{\infty} \frac{Q(s, kx)}{k^s}$$

where $P(a, x), Q(a, x)$ is the incomplete gamma function ratios. The Debye functions are:

$$D_n(x) = \frac{n}{x^n} \int_0^x \frac{t^n}{e^t - 1} dt = \Gamma(n+1) \zeta_x(n+1)$$

and

$$\int_0^x \frac{t^n}{e^t - 1} dt = \Gamma(n+1) \zeta_x(n+1)$$

Todo : We should return both the Debye function and it's complement.

Compute the Debye function:

$$D_n(x) = 1 - \sum_{k=1}^{\infty} e^{-kx} \frac{n}{k} \sum_{m=0}^n \frac{n!}{(n-m)!} \frac{1}{(kx)^m}$$

Abramowitz & Stegun 27.1.2

Compute the Debye function:

$$D_n(x) = 1 - \frac{nx}{2(n+1)} + n \sum_{k=1}^{\infty} \frac{B_{2k} x^{2k}}{(2k+n)(2k)!}$$

for $|x| < 2\pi$. Abramowitz-Stegun 27.1.1

Definition at line 818 of file `sf_zeta.tcc`.

9.3.1.78 `template<typename _Tp > void std::__detail::__debye_region (std::complex<_Tp > __alpha, int & __indexr, char & __aorb)`

Compute the Debye region in the complex plane.

Definition at line 54 of file `sf_hankel.tcc`.

Referenced by `__hankel()`.

9.3.1.79 `template<typename _Tp> _Tp std::__detail::__dilog (_Tp __x)`

Compute the dilogarithm function $Li_2(x)$ by summation for $x \leq 1$.

The dilogarithm function is defined by:

$$Li_2(x) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For $|x|$ near 1 use the reflection formulae:

$$Li_2(-x) + Li_2(1-x) = \frac{\pi^2}{6} - \ln(x) \ln(1-x)$$

$$Li_2(-x) - Li_2(1-x) - \frac{1}{2}Li_2(1-x^2) = -\frac{\pi^2}{12} - \ln(x) \ln(1-x)$$

For $x < 1$ use the reflection formula:

$$Li_2(1-x) - Li_2\left(1 - \frac{1}{1-x}\right) - \frac{1}{2}(\ln(x))^2$$

Definition at line 196 of file `sf_zeta.tcc`.

9.3.1.80 `template<typename _Tp> _Tp std::__detail::__dirichlet_beta (std::complex<_Tp> __s)`

Return the Dirichlet beta function. Currently, s must be real (complex type but negligible imaginary part.) Otherwise `std::domain_error` is thrown. The Dirichlet beta function, in terms of the polylogarithm, is

$$\beta(s) = \text{Im } Li_s(i)$$

Parameters

<code>__s</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 1156 of file `sf_polylog.tcc`.

References `__polylog()`.

9.3.1.81 `template<typename _Tp > _Tp std::__detail::__dirichlet_beta (_Tp __s)`

Return the Dirichlet beta function for real argument. The Dirichlet beta function, in terms of the polylogarithm, is

$$\beta(s) = \operatorname{Im} Li_s(i)$$

Parameters

<code>__s</code>	The real argument.
------------------	--------------------

Returns

The Dirichlet Beta function of real argument.

Definition at line 1181 of file `sf_polylog.tcc`.

References `__polylog()`.

9.3.1.82 `template<typename _Tp > std::complex<_Tp> std::__detail::__dirichlet_eta (std::complex<_Tp > __s)`

Return the Dirichlet eta function. Currently, `s` must be real (complex type but negligible imaginary part.) Otherwise `std::domain_error` is thrown. The Dirichlet eta function, in terms of the polylogarithm, is

$$\eta(s) = -\operatorname{Re} Li_s(-1)$$

Parameters

<code>__s</code>	The complex (but on-real-axis) argument.
------------------	--

Returns

The complex Dirichlet eta function.

Exceptions

<code>std::domain_error</code>	if the argument has a significant imaginary part.
--------------------------------	---

Definition at line 1092 of file `sf_polylog.tcc`.

References `__polylog()`.

Referenced by `__dirichlet_eta()`, and `__dirichlet_lambda()`.

9.3.1.83 `template<typename _Tp> _Tp std::__detail::__dirichlet_eta (_Tp __s)`

Return the Dirichlet eta function for real argument. The Dirichlet eta function, in terms of the polylogarithm, is

$$\eta(s) = -\operatorname{Re} Li_s(-1)$$

Parameters

<code>__s</code>	The real argument.
------------------	--------------------

Returns

The Dirichlet eta function.

Definition at line 1116 of file sf_polylog.tcc.

References `__dirichlet_eta()`, `__gnu_cxx::__fp_is_integer()`, `__gamma()`, `__polylog()`, and `__sin_pi()`.

9.3.1.84 `template<typename _Tp> _Tp std::__detail::__dirichlet_lambda (_Tp __s)`

Return the Dirichlet lambda function for real argument.

$$\lambda(s) = \frac{1}{2}(\zeta(s) + \eta(s))$$

Parameters

<code>__s</code>	The real argument.
------------------	--------------------

Returns

The Dirichlet lambda function.

Definition at line 1201 of file sf_polylog.tcc.

References `__dirichlet_eta()`, and `__riemann_zeta()`.

9.3.1.85 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__double_factorial (int __n)`

Return the double factorial of the integer n.

The double factorial is defined for integral n by:

$$n!! = 135\dots(n-2)n, \text{ odd } n!! = 246\dots(n-2)n, \text{ even } -1!! = 10!! = 1$$

The double factorial is defined for odd negative integers in the obvious way:

$$(-2m-1)!! = 1/(1(-1)(-3)\dots(-2m+1)(-2m-1)) = \frac{(-1)^m}{(2m-1)!!}$$

for $n = -2m - 1$.

Definition at line 1673 of file sf_gamma.tcc.

References std::__detail::_Factorial_table<_Tp>::_factorial, __log_double_factorial(), std::__detail::_Factorial_table<_Tp>::_n, _S_double_factorial_table, and _S_neg_double_factorial_table.

9.3.1.86 template<typename _Tp> _Tp std::__detail::__ellint_1 (_Tp __k, _Tp __phi)

Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.

The incomplete elliptic integral of the first kind is defined as

$$F(k, \phi) = \int_0^\phi \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}}$$

Parameters

<code>__k</code>	The argument of the elliptic function.
<code>__phi</code>	The integral limit argument of the elliptic function.

Returns

The elliptic function of the first kind.

Definition at line 597 of file sf_ellint.tcc.

References __comp_ellint_1(), and __ellint_rf().

Referenced by __heuman_lambda().

9.3.1.87 template<typename _Tp> _Tp std::__detail::__ellint_2 (_Tp __k, _Tp __phi)

Return the incomplete elliptic integral of the second kind $E(k, \phi)$ using the Carlson formulation.

The incomplete elliptic integral of the second kind is defined as

$$E(k, \phi) = \int_0^\phi \sqrt{1 - k^2 \sin^2 \theta}$$

Parameters

<code>__k</code>	The argument of the elliptic function.
<code>__phi</code>	The integral limit argument of the elliptic function.

Returns

The elliptic function of the second kind.

Definition at line 678 of file sf_ellint.tcc.

References `__comp_ellint_2()`, `__ellint_rd()`, and `__ellint_rf()`.

9.3.1.88 `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 773 of file sf_ellint.tcc.

References `__comp_ellint_3()`, `__ellint_rf()`, and `__ellint_rj()`.

9.3.1.89 `template<typename _Tp> _Tp std::__detail::__ellint_cel(_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`

Return the Bulirsch complete elliptic integrals.

Definition at line 928 of file sf_ellint.tcc.

References `__ellint_rf()`, and `__ellint_rj()`.

9.3.1.90 `template<typename _Tp> _Tp std::__detail::__ellint_d (_Tp __k, _Tp __phi)`

Return the Legendre elliptic integral D.

Definition at line 814 of file `sf_ellint.tcc`.

References `__ellint_rd()`.

9.3.1.91 `template<typename _Tp> _Tp std::__detail::__ellint_el1 (_Tp __x, _Tp __k_c)`

Return the Bulirsch elliptic integrals of the first kind.

Definition at line 856 of file `sf_ellint.tcc`.

References `__ellint_rf()`.

9.3.1.92 `template<typename _Tp> _Tp std::__detail::__ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`

Return the Bulirsch elliptic integrals of the second kind.

Definition at line 877 of file `sf_ellint.tcc`.

References `__ellint_rd()`, and `__ellint_rf()`.

9.3.1.93 `template<typename _Tp> _Tp std::__detail::__ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`

Return the Bulirsch elliptic integrals of the third kind.

Definition at line 902 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

9.3.1.94 `template<typename _Tp> _Tp std::__detail::__ellint_rc (_Tp __x, _Tp __y)`

Return the Carlson elliptic function $R_C(x, y) = R_F(x, y, y)$ where $R_F(x, y, z)$ is the Carlson elliptic function of the first kind.

The Carlson elliptic function is defined by:

$$R_C(x, y) = \frac{1}{2} \int_0^\infty \frac{dt}{(t+x)^{1/2}(t+y)}$$

Based on Carlson's algorithms:

- B. C. Carlson Numer. Math. 33, 1 (1979)
- B. C. Carlson, Special Functions of Applied Mathematics (1977)
- Numerical Recipes in C, 2nd ed, pp. 261-269, by Press, Teukolsky, Vetterling, Flannery (1992)

Parameters

$_x$	The first argument.
$_y$	The second argument.

Returns

The Carlson elliptic function.

Definition at line 84 of file sf_ellint.tcc.

Referenced by `__ellint_rf()`, and `__ellint_rj()`.

9.3.1.95 `template<typename _Tp> _Tp std::__detail::__ellint_rd (_Tp __x, _Tp __y, _Tp __z)`

Return the Carlson elliptic function of the second kind $R_D(x, y, z) = R_J(x, y, z, z)$ where $R_J(x, y, z, p)$ is the Carlson elliptic function of the third kind.

The Carlson elliptic function of the second kind is defined by:

$$R_D(x, y, z) = \frac{3}{2} \int_0^\infty \frac{dt}{(t+x)^{1/2}(t+y)^{1/2}(t+z)^{3/2}}$$

Based on Carlson's algorithms:

- B. C. Carlson Numer. Math. 33, 1 (1979)
- B. C. Carlson, Special Functions of Applied Mathematics (1977)
- Numerical Recipes in C, 2nd ed, pp. 261-269, by Press, Teukolsky, Vetterling, Flannery (1992)

Parameters

$_x$	The first of two symmetric arguments.
$_y$	The second of two symmetric arguments.
$_z$	The third argument.

Returns

The Carlson elliptic function of the second kind.

Definition at line 166 of file sf_ellint.tcc.

Referenced by `__comp_ellint_2()`, `__comp_ellint_d()`, `__ellint_2()`, `__ellint_d()`, `__ellint_el2()`, `__ellint_rg()`, and `__ellint_rj()`.

9.3.1.96 `template<typename _Tp> _Tp std::__detail::__ellint_rf (_Tp __x, _Tp __y, _Tp __z)`

Return the Carlson elliptic function $R_F(x, y, z)$ of the first kind.

The Carlson elliptic function of the first kind is defined by:

$$R_F(x, y, z) = \frac{1}{2} \int_0^\infty \frac{dt}{(t+x)^{1/2}(t+y)^{1/2}(t+z)^{1/2}}$$

Parameters

<code>__x</code>	The first of three symmetric arguments.
<code>__y</code>	The second of three symmetric arguments.
<code>__z</code>	The third of three symmetric arguments.

Returns

The Carlson elliptic function of the first kind.

Definition at line 280 of file `sf_ellint.tcc`.

References `__comp_ellint_rf()`, and `__ellint_rc()`.

Referenced by `__comp_ellint_2()`, `__comp_ellint_3()`, `__ellint_1()`, `__ellint_2()`, `__ellint_3()`, `__ellint_cel()`, `__ellint_el1()`, `__ellint_el2()`, `__ellint_el3()`, and `__heuman_lambda()`.

9.3.1.97 `template<typename _Tp> _Tp std::__detail::__ellint_rg (_Tp __x, _Tp __y, _Tp __z)`

Return the symmetric Carlson elliptic function of the second kind $R_G(x, y, z)$.

The Carlson symmetric elliptic function of the second kind is defined by:

$$R_G(x, y, z) = \frac{1}{4} \int_0^\infty dt [(t+x)(t+y)(t+z)]^{-1/2} \left(\frac{x}{t+x} + \frac{y}{t+y} + \frac{z}{t+z} \right)$$

Based on Carlson's algorithms:

- B. C. Carlson Numer. Math. 33, 1 (1979)
- B. C. Carlson, Special Functions of Applied Mathematics (1977)
- Numerical Recipes in C, 2nd ed, pp. 261-269, by Press, Teukolsky, Vetterling, Flannery (1992)

Parameters

$_x$	The first of three symmetric arguments.
$_y$	The second of three symmetric arguments.
$_z$	The third of three symmetric arguments.

Returns

The Carlson symmetric elliptic function of the second kind.

Definition at line 411 of file sf_ellint.tcc.

References `__comp_ellint_rg()`, and `__ellint_rd()`.

9.3.1.98 `template<typename _Tp> _Tp std::__detail::__ellint_rj(_Tp __x, _Tp __y, _Tp __z, _Tp __p)`

Return the Carlson elliptic function $R_J(x, y, z, p)$ of the third kind.

The Carlson elliptic function of the third kind is defined by:

$$R_J(x, y, z, p) = \frac{3}{2} \int_0^\infty \frac{dt}{(t+x)^{1/2}(t+y)^{1/2}(t+z)^{1/2}(t+p)}$$

Based on Carlson's algorithms:

- B. C. Carlson Numer. Math. 33, 1 (1979)
- B. C. Carlson, Special Functions of Applied Mathematics (1977)
- Numerical Recipes in C, 2nd ed, pp. 261-269, by Press, Teukolsky, Vetterling, Flannery (1992)

Parameters

$_x$	The first of three symmetric arguments.
$_y$	The second of three symmetric arguments.
$_z$	The third of three symmetric arguments.
$_p$	The fourth argument.

Returns

The Carlson elliptic function of the fourth kind.

Definition at line 459 of file sf_ellint.tcc.

References `__ellint_rc()`, and `__ellint_rd()`.

Referenced by `__comp_ellint_3()`, `__ellint_3()`, `__ellint_cel()`, `__ellint_el3()`, `__heuman_lambda()`, and `__jacobi_zeta()`.

9.3.1.99 `template<typename _Tp> _Tp std::__detail::__ellnome (_Tp __k)`

Return the elliptic nome given the modulus k .

$$q(k) = \exp\left(-\pi \frac{K(k')}{K(k)}\right)$$

Definition at line 307 of file sf_theta.tcc.

References `__ellnome_k()`, and `__ellnome_series()`.

Referenced by `__theta_c()`, `__theta_d()`, `__theta_n()`, and `__theta_s()`.

9.3.1.100 `template<typename _Tp> _Tp std::__detail::__ellnome_k (_Tp __k)`

Use the arithmetic-geometric mean to calculate the elliptic nome given the elliptic argument k .

$$q(k) = \exp\left(-\pi \frac{K(k')}{K(k)}\right)$$

where $k' = \sqrt{1 - k^2}$ is the complementary elliptic argument and K is the Legendre elliptic integral of the first kind.

Definition at line 290 of file sf_theta.tcc.

References `__comp_ellint_1()`.

Referenced by `__ellnome()`.

9.3.1.101 `template<typename _Tp> _Tp std::__detail::__ellnome_series (_Tp __k)`

Use MacLaurin series to calculate the elliptic nome given the elliptic argument k .

$$q(k) = \exp\left(-\pi \frac{K(k')}{K(k)}\right)$$

where $k' = \sqrt{1 - k^2}$ is the complementary elliptic argument and K is the Legendre elliptic integral of the first kind.

Definition at line 269 of file sf_theta.tcc.

Referenced by `__ellnome()`.

9.3.1.102 `template<typename _Tp> _Tp std::__detail::__euler (unsigned int __n) [inline]`

This returns Euler number E_n .

Parameters

$_n$	the order n of the Euler number.
-------	----------------------------------

Returns

The Euler number of order n.

Definition at line 119 of file sf_euler.tcc.

9.3.1.103 `template<typename _Tp> _Tp std::__detail::__euler (unsigned int __n, _Tp __x)`

Return the Euler polynomial $E_n(x)$ of order n at argument x.

The derivative is proportional to the previous polynomial:

$$E'_n(x) = nE_{n-1}(x)$$

$$E_n(1/2) = \frac{E_n}{2^n}, \text{ where } E_n \text{ is the n-th Euler number.}$$

Definition at line 137 of file sf_euler.tcc.

References `__bernoulli()`.

9.3.1.104 `template<typename _Tp> _Tp std::__detail::__euler_series (unsigned int __n)`

Return the Euler number from lookup or by series expansion.

The Euler numbers are given by the recursive sum:

$$E_n = B_n(1) = B_n$$

where $E_0 = 1$, $E_1 = 0$, $E_2 = -1$

Todo Find a way to predict the maximum Euler number for a type.

Definition at line 61 of file sf_euler.tcc.

9.3.1.105 `template<typename _Tp> _Tp std::__detail::__eulerian_1_recur (unsigned int __n, unsigned int __m)`

Return the Eulerian number of the first kind. The Eulerian numbers of the first kind are defined by recursion:

$$\left\langle \begin{matrix} n \\ m \end{matrix} \right\rangle = (n-m) \left\langle \begin{matrix} n-1 \\ m-1 \end{matrix} \right\rangle + (m+1) \left\langle \begin{matrix} n-1 \\ m \end{matrix} \right\rangle \text{ for } n > 0$$

Note that $A(n, m)$ is a common older notation.

Definition at line 166 of file sf_euler.tcc.

9.3.1.106 `template<typename _Tp> _Tp std::__detail::__eulerian_2 (unsigned int __n, unsigned int __m) [inline]`

Return the Eulerian number of the second kind. The Eulerian numbers of the second kind are defined by recursion:

$$A(n, m) = (2n - m - 1)A(n - 1, m - 1) + (m + 1)A(n - 1, m) \text{ for } n > 0$$

Definition at line 254 of file sf_euler.tcc.

9.3.1.107 `template<typename _Tp> _Tp std::__detail::__eulerian_2_recur (unsigned int __n, unsigned int __m)`

Return the Eulerian number of the first kind. The Eulerian numbers of the first kind are defined by recursion:

$$\left\langle n \atop m \right\rangle = (n - m) \left\langle n - 1 \atop m - 1 \right\rangle + (m + 1) \left\langle n - 1 \atop m \right\rangle \text{ for } n > 0$$

Note that 2f\$ A(n,m)

Definition at line 219 of file sf_euler.tcc.

9.3.1.108 `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

<code>__n</code>	The order of the exponential integral function.
<code>__x</code>	The argument of the exponential integral function.

Returns

The exponential integral.

Todo Study arbitrary switch to large-n $E_n(x)$.

Todo Find a good asymptotic switch point in $E_n(x)$.

Definition at line 476 of file sf_expint.tcc.

References `__expint_E1()`, `__expint_En_asymp()`, `__expint_En_cont_frac()`, `__expint_En_large_n()`, and `__expint_En_series()`.

Referenced by `__logint()`.

9.3.1.109 `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

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Returns

The exponential integral.

Definition at line 517 of file sf_expint.tcc.

References `__expint_Ei()`.

9.3.1.110 `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

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Returns

The exponential integral.

Todo Find a good asymptotic switch point in $E_1(x)$.

Todo Find a good asymptotic switch point in $E_1(x)$.

Definition at line 381 of file sf_expint.tcc.

References `__expint_E1_asymp()`, `__expint_E1_series()`, `__expint_Ei()`, and `__expint_En_cont_frac()`.

Referenced by `__coshint()`, `__expint()`, `__expint_Ei()`, `__expint_En_recursion()`, and `__sinhint()`.

9.3.1.111 `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

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Returns

The exponential integral.

Definition at line 114 of file `sf_expint.tcc`.

Referenced by `__expint_E1()`.

9.3.1.112 `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

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Returns

The exponential integral.

Definition at line 76 of file `sf_expint.tcc`.

Referenced by `__expint_E1()`.

9.3.1.113 `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 356 of file sf_expint.tcc.

References `__expint_E1()`, `__expint_Ei_asyp()`, and `__expint_Ei_series()`.

Referenced by `__coshint()`, `__expint()`, `__expint_E1()`, and `__sinhint()`.

9.3.1.114 `template<typename _Tp> _Tp std::__detail::__expint_Ei_asyp (_Tp __x)`

Return the exponential integral $Ei(x)$ by asymptotic expansion.

The exponential integral is given by

$$Ei(x) = - \int_{-x}^{\infty} \frac{e^t}{t} dt$$

Parameters

\leftrightarrow	The argument of the exponential integral function.
x	

Returns

The exponential integral.

Definition at line 322 of file sf_expint.tcc.

Referenced by `__expint_Ei()`.

9.3.1.115 `template<typename _Tp> _Tp std::__detail::__expint_Ei_series (_Tp __x)`

Return the exponential integral $Ei(x)$ by series summation.

The exponential integral is given by

$$Ei(x) = - \int_{-x}^{\infty} \frac{e^t}{t} dt$$

Parameters

\leftrightarrow __x	The argument of the exponential integral function.
--------------------------	--

Returns

The exponential integral.

Definition at line 289 of file sf_expint.tcc.

Referenced by __expint_Ei().

9.3.1.116 `template<typename _Tp> _Tp std::__detail::__expint_En_asymp (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ for large argument.

The exponential integral is given by

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

Parameters

\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 410 of file sf_expint.tcc.

Referenced by __expint().

9.3.1.117 `template<typename _Tp> _Tp std::__detail::__expint_En_cont_frac (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by continued fractions.

The exponential integral is given by

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

Parameters

\leftrightarrow _n	The order of the exponential integral function.
\leftrightarrow _x	The argument of the exponential integral function.

Returns

The exponential integral.

Definition at line 198 of file sf_expint.tcc.

Referenced by __expint(), and __expint_E1().

9.3.1.118 `template<typename _Tp> _Tp std::__detail::__expint_En_large_n (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ for large order.

The exponential integral is given by

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

Parameters

\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 442 of file sf_expint.tcc.

Referenced by __expint().

9.3.1.119 `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 244 of file sf_expint.tcc.

References __expint_E1().

9.3.1.120 `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 150 of file sf_expint.tcc.

References __psi().

Referenced by __expint().

9.3.1.121 `template<typename _Tp> _Tp std::__detail::__exponential_cdf (_Tp __lambda, _Tp __x)`

Return the exponential cumulative probability density function.

The formula for the exponential cumulative probability density function is

$$F(x|\lambda) = 1 - e^{-\lambda x} \text{ for } x \geq 0$$

Definition at line 328 of file sf_distributions.tcc.

9.3.1.122 `template<typename _Tp> _Tp std::__detail::__exponential_cdfc (_Tp __lambda, _Tp __x)`

Return the complement of the exponential cumulative probability density function.

The formula for the complement of the exponential cumulative probability density function is

$$F(x|\lambda) = e^{-\lambda x} \text{ for } x \geq 0$$

Definition at line 350 of file sf_distributions.tcc.

9.3.1.123 `template<typename _Tp> _Tp std::__detail::__exponential_pdf (_Tp __lambda, _Tp __x)`

Return the exponential probability density function.

The formula for the exponential probability density function is

$$f(x|\lambda) = \lambda e^{-\lambda x} \text{ for } x \geq 0$$

Definition at line 308 of file sf_distributions.tcc.

9.3.1.124 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (unsigned int __n)`

Return the factorial of the integer n.

The factorial is:

$$n! = 12...(n-1)n, 0! = 1$$

Definition at line 1615 of file sf_gamma.tcc.

References `std::__detail::__Factorial_table<_Tp>::__n`, and `_S_factorial_table`.

9.3.1.125 `template<typename _Tp> _Tp std::__detail::__falling_factorial (_Tp __a, int __n)`

Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n . The falling factorial function is defined by

$$a^{\underline{n}} = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $n^{\underline{n}} = n!$.

Definition at line 2903 of file `sf_gamma.tcc`.

References `__gnu_cxx::__fp_is_integer()`, `__log_gamma()`, `__log_gamma_sign()`, and `std::__detail::__Factorial_table<_Tp>::__n`.

Referenced by `__falling_factorial()`, and `__log_falling_factorial()`.

9.3.1.126 `template<typename _Tp> _Tp std::__detail::__falling_factorial (_Tp __a, _Tp __nu)`

Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and order ν . The falling factorial function is defined by

$$a^{\underline{\nu}} = \Gamma(a + 1) / \Gamma(a - \nu + 1)$$

.

Definition at line 2958 of file `sf_gamma.tcc`.

References `__falling_factorial()`, `__gnu_cxx::__fp_is_integer()`, `__log_gamma()`, and `__log_gamma_sign()`.

9.3.1.127 `template<typename _Sp, typename _Tp> _Tp std::__detail::__fermi_dirac (_Sp __s, _Tp __x)`

Return the Fermi-Dirac integral of integer or real order s and real argument x .

See also

https://en.wikipedia.org/wiki/Clausen_function
<http://dlmf.nist.gov/25.12.16>

$$F_s(x) = \frac{1}{\Gamma(s + 1)} \int_0^\infty \frac{t^s}{e^{t-x} + 1} dt = -Li_{s+1}(-e^x)$$

Parameters

<code>__s</code>	The order $s > -1$.
<code>__x</code>	The real argument.

Returns

The real Fermi-Dirac cosine sum $F_s(x)$,

Definition at line 1392 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.1.128 `template<typename _Tp> _Tp std::__detail::__fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .

The f-distribution propability function is related to the incomplete beta function:

$$Q(F|\nu_1, \nu_2) = I_{\frac{\nu_2}{\nu_2 + \nu_1 F}}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right)$$

Parameters

<code>__nu1</code>	The number of degrees of freedom of sample 1
<code>__nu2</code>	The number of degrees of freedom of sample 2
<code>__F</code>	The F statistic

Definition at line 523 of file sf_distributions.tcc.

References `__beta_inc()`.

9.3.1.129 `template<typename _Tp> _Tp std::__detail::__fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)`

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .

The f-distribution propability function is related to the incomplete beta function:

$$P(F|\nu_1, \nu_2) = 1 - I_{\frac{\nu_2}{\nu_2 + \nu_1 F}}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right) = 1 - Q(F|\nu_1, \nu_2)$$

Parameters

<code>__F</code>	
<code>__nu1</code>	
<code>__nu2</code>	

Definition at line 552 of file sf_distributions.tcc.

References `__beta_inc()`.

9.3.1.130 `template<typename _Tp> _Tp std::__detail::__fisher_f_pdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .

The f-distribution propability function is related to the incomplete beta function:

$$Q(F|\nu_1, \nu_2) = I_{\frac{\nu_2}{\nu_2 + \nu_1 F}}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right)$$

Parameters

<code>__nu1</code>	The number of degrees of freedom of sample 1
<code>__nu2</code>	The number of degrees of freedom of sample 2
<code>__F</code>	The F statistic

Definition at line 493 of file `sf_distributions.tcc`.

References `__beta()`.

9.3.1.131 `template<typename _Tp> __gnu_cxx::__fock_airy_t<_Tp, std::complex<_Tp>> std::__detail::__fock_airy (_Tp __x)`

Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w'_1(x)$ and $w'_2(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$

.

Parameters

<code>__x</code>	The argument of the Airy functions.
------------------	-------------------------------------

Returns

A struct containing the Fock-type Airy functions of the first and second kinds and their derivatives.

Definition at line 549 of file `sf_mod_bessel.tcc`.

References `__airy()`.

9.3.1.132 `template<typename _Tp> std::complex<_Tp> std::__detail::__fresnel (const _Tp __x)`

Return the Fresnel cosine and sine integrals as a complex number $\$f[C(x) + iS(x) \$f]$.

The Fresnel cosine integral is defined by:

$$C(x) = \int_0^x \cos\left(\frac{\pi}{2}t^2\right)dt$$

The Fresnel sine integral is defined by:

$$S(x) = \int_0^x \sin\left(\frac{\pi}{2}t^2\right)dt$$

Parameters

\leftrightarrow	The argument
x	

Definition at line 170 of file sf_fresnel.tcc.

References `__fresnel_cont_frac()`, and `__fresnel_series()`.

9.3.1.133 `template<typename _Tp> void std::__detail::__fresnel_cont_frac (const _Tp __ax, _Tp & __Cf, _Tp & __Sf)`

This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.

Definition at line 109 of file sf_fresnel.tcc.

Referenced by `__fresnel()`.

9.3.1.134 `template<typename _Tp> void std::__detail::__fresnel_series (const _Tp __ax, _Tp & __Cf, _Tp & __Sf)`

This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.

Definition at line 51 of file sf_fresnel.tcc.

Referenced by `__fresnel()`.

9.3.1.135 `template<typename _Tp> _Tp std::__detail::__gamma (_Tp __a)`

Return the gamma function $\Gamma(a)$. The gamma function is defined by:

$$\Gamma(a) = \int_0^\infty e^{-t} t^{a-1} dt (a > 0)$$

.

Parameters

\leftrightarrow	The argument of the gamma function.
a	

Returns

The gamma function.

Definition at line 2601 of file sf_gamma.tcc.

References `__gnu_cxx::__fp_is_integer()`, `__gamma_reciprocal_series()`, `__log_gamma()`, `__log_gamma_sign()`, `std::__detail::_Factorial_table<_Tp>::__n`, and `_S_factorial_table`.

Referenced by `__beta_gamma()`, `__binomial()`, `__dirichlet_eta()`, `__gamma_cdf()`, `__gamma_cdfc()`, `__gamma_pdf()`, `__gamma_reciprocal()`, `__gamma_reciprocal_series()`, `__hurwitz_zeta_polylog()`, `__polylog_exp_pos()`, `__riemann_zeta()`, `__riemann_zeta_glob()`, `__riemann_zeta_m_1()`, `__riemann_zeta_sum()`, `__student_t_pdf()`, and `std::__detail::__Airy_series<_Tp>::__S_Scorer2()`.

9.3.1.136 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma (_Tp __a, _Tp __x)`

Return the incomplete gamma functions.

Definition at line 2728 of file sf_gamma.tcc.

References `__gnu_cxx::__fp_is_integer()`, `__gamma_cont_frac()`, and `__gamma_series()`.

9.3.1.137 `template<typename _Tp> _Tp std::__detail::__gamma_cdf (_Tp __alpha, _Tp __beta, _Tp __x)`

Return the gamma cumulative propability distribution function.

The formula for the gamma probability density function is:

$$\Gamma(x|\alpha, \beta) = \frac{1}{\beta\Gamma(\alpha)} (x/\beta)^{\alpha-1} e^{-x/\beta}$$

Definition at line 141 of file sf_distributions.tcc.

References `__gamma()`, and `__tgamma_lower()`.

9.3.1.138 `template<typename _Tp> _Tp std::__detail::__gamma_cdfc (_Tp __alpha, _Tp __beta, _Tp __x)`

Return the gamma complementary cumulative propability distribution function.

The formula for the gamma probability density function is:

$$\Gamma(x|\alpha, \beta) = \frac{1}{\beta\Gamma(\alpha)} (x/\beta)^{\alpha-1} e^{-x/\beta}$$

Definition at line 162 of file sf_distributions.tcc.

References `__gamma()`, and `__tgamma()`.

9.3.1.139 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma_cont_frac (_Tp __a, _Tp __x)`

Return the incomplete gamma function by continued fraction.

Definition at line 2683 of file `sf_gamma.tcc`.

References `__log_gamma()`, `__log_gamma_sign()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__gamma()`, `__pgamma()`, `__qgamma()`, `__tgamma()`, and `__tgamma_lower()`.

9.3.1.140 `template<typename _Tp> _Tp std::__detail::__gamma_pdf (_Tp __alpha, _Tp __beta, _Tp __x)`

Return the gamma propability distribution function.

The formula for the gamma probability density function is:

$$\Gamma(x|\alpha, \beta) = \frac{1}{\beta\Gamma(\alpha)} (x/\beta)^{\alpha-1} e^{-x/\beta}$$

Definition at line 121 of file `sf_distributions.tcc`.

References `__gamma()`.

9.3.1.141 `template<typename _Tp> _Tp std::__detail::__gamma_reciprocal (_Tp __a)`

Return the reciprocal of the Gamma function:

$$\frac{1}{\Gamma(a)}$$

Parameters

<code>↔ __a</code>	The argument of the reciprocal of the gamma function.
------------------------	---

Returns

The reciprocal of the gamma function.

Definition at line 2246 of file `sf_gamma.tcc`.

References `std::__detail::_Factorial_table<_Tp>::__factorial`, `__gnu_cxx::__fp_is_integer()`, `__gamma()`, `__gamma↔
_reciprocal_series()`, `std::__detail::_Factorial_table<_Tp>::__n`, `__sin_pi()`, and `_S_factorial_table`.

Referenced by `__polylog_exp_asymp()`.

9.3.1.142 `template<typename _Tp> _Tp std::__detail::__gamma_reciprocal_series (_Tp __a)`

Return the reciprocal of the Gamma function by series. The reciprocal of the Gamma function is given by

$$\frac{1}{\Gamma(a)} = \sum_{k=1}^{\infty} c_k a^k$$

where the coefficients are defined by recursion:

$$c_{k+1} = \frac{1}{k} \left[\gamma_E c_k + (-1)^k \sum_{j=1}^{k-1} (-1)^j \zeta(j+1-k) c_j \right]$$

where $c_1 = 1$

Parameters

<code>__a</code>	The argument of the reciprocal of the gamma function.
------------------	---

Returns

The reciprocal of the gamma function.

Definition at line 2180 of file `sf_gamma.tcc`.

References `__gamma()`.

Referenced by `__gamma()`, `__gamma_reciprocal()`, and `__gamma_temme()`.

9.3.1.143 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma_series (_Tp __a, _Tp __x)`

Return the incomplete gamma function by series summation.

$$\gamma(a, x) = x^a e^{-x} \sum_{k=1}^{\infty} \frac{x^k}{(a)_k}$$

.

Definition at line 2638 of file `sf_gamma.tcc`.

References `__gnu_cxx::__fp_is_integer()`, `__log_gamma()`, `__log_gamma_sign()`, and `std::__detail::__Factorial_table<_Tp>::__n`.

Referenced by `__gamma()`, `__pgamma()`, `__qgamma()`, `__tgamma()`, and `__tgamma_lower()`.

9.3.1.144 `template<typename _Tp> __gnu_cxx::__gamma_temme_t<_Tp> std::__detail::__gamma_temme (_Tp __mu)`

Compute the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ 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.
-------------------	---

Returns

An output structure containing four gamma functions.

Definition at line 158 of file `sf_bessel.tcc`.

References `__gamma_reciprocal_series()`.

Referenced by `__cyl_bessel_ik_steel()`, and `__cyl_bessel_jn_steel()`.

9.3.1.145 `template<typename _Tp > _Tp std::__detail::__gauss (_Tp __x)`

The CDF of the normal distribution. i.e. the integrated lower tail of the normal PDF.

Definition at line 70 of file `sf_owens.t.tcc`.

9.3.1.146 `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 63 of file `sf_gegenbauer.tcc`.

9.3.1.147 `template<typename _Tp> __gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>, std::complex<_Tp>> std::__detail::__hankel (std::complex<_Tp> __nu, std::complex<_Tp> __z)`

Parameters

in	<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.

Returns

A struct containing the cylindrical Hankel functions of the first and second kinds and their derivatives.

Definition at line 1081 of file `sf_hankel.tcc`.

References `__debye_region()`, `__hankel_debye()`, and `__hankel_uniform()`.

Referenced by `__cyl_bessel()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann()`, and `__sph_hankel()`.

9.3.1.148 `template<typename _Tp> __gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>, std::complex<_Tp>> std::__detail::__hankel_debye (std::complex<_Tp> __nu, std::complex<_Tp> __z, std::complex<_Tp> __alpha, int __indexr, char &__aorb, int &__morn)`

Parameters

in	<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>	

Returns

A struct containing the cylindrical Hankel functions of the first and second kinds and their derivatives.

Definition at line 914 of file `sf_hankel.tcc`.

References `__sin_pi()`.

Referenced by `__hankel()`.

9.3.1.149 `template<typename _Tp> void std::__detail::__hankel_params (std::complex<_Tp> __nu, std::complex<_Tp> __zhat, std::complex<_Tp> &__p, std::complex<_Tp> &__p2, std::complex<_Tp> &__nup2, std::complex<_Tp> &__num2, std::complex<_Tp> &__num1d3, std::complex<_Tp> &__num2d3, std::complex<_Tp> &__num4d3, std::complex<_Tp> &__zeta, std::complex<_Tp> &__zetaphf, std::complex<_Tp> &__zetamhf, std::complex<_Tp> &__zetam3hf, std::complex<_Tp> &__zetrat)`

Compute parameters depending on `z` and `nu` that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.

Definition at line 109 of file sf_hankel.tcc.

Referenced by `__hankel_uniform_outer()`.

```
9.3.1.150 template<typename _Tp> __gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>,
std::complex<_Tp>> std::__detail::__hankel_uniform ( std::complex<_Tp> __nu, std::complex<_Tp> __z )
```

This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.

Parameters

in	<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.

Returns

A struct containing the cylindrical Hankel functions of the first and second kinds and their derivatives.

Definition at line 861 of file sf_hankel.tcc.

References `__hankel_uniform_olver()`.

Referenced by `__hankel()`.

```
9.3.1.151 template<typename _Tp> __gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>,
std::complex<_Tp>> std::__detail::__hankel_uniform_olver ( std::complex<_Tp> __nu, std::complex<_Tp> __z )
```

Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order `nu` along with their derivatives.

Parameters

in	<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.

Returns

A struct containing the cylindrical Hankel functions of the first and second kinds and their derivatives.

Definition at line 778 of file sf_hankel.tcc.

References `__hankel_uniform_outer()`, and `__hankel_uniform_sum()`.

Referenced by `__hankel_uniform()`.

9.3.1.152 `template<typename _Tp> void std::__detail::__hankel_uniform_outer (std::complex<_Tp> __nu, std::complex<_Tp> __z, _Tp __eps, std::complex<_Tp> & __zhat, std::complex<_Tp> & __1dnsq, std::complex<_Tp> & __num1d3, std::complex<_Tp> & __num2d3, std::complex<_Tp> & __p, std::complex<_Tp> & __p2, std::complex<_Tp> & __etm3h, std::complex<_Tp> & __etrat, std::complex<_Tp> & __Aip, std::complex<_Tp> & __o4dp, std::complex<_Tp> & __Aim, std::complex<_Tp> & __o4dm, std::complex<_Tp> & __od2p, std::complex<_Tp> & __od0dp, std::complex<_Tp> & __od2m, std::complex<_Tp> & __od0dm)`

Compute outer factors and associated functions of `z` and `nu` appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of `z` and `nu` returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.

Definition at line 248 of file `sf_hankel.tcc`.

References `__airy_arg()`, and `__hankel_params()`.

Referenced by `__hankel_uniform_olver()`.

9.3.1.153 `template<typename _Tp> void std::__detail::__hankel_uniform_sum (std::complex<_Tp> __p, std::complex<_Tp> __p2, std::complex<_Tp> __num2, std::complex<_Tp> __zetam3hf, std::complex<_Tp> __Aip, std::complex<_Tp> __o4dp, std::complex<_Tp> __Aim, std::complex<_Tp> __o4dm, std::complex<_Tp> __od2p, std::complex<_Tp> __od0dp, std::complex<_Tp> __od2m, std::complex<_Tp> __od0dm, _Tp __eps, std::complex<_Tp> & __H1sum, std::complex<_Tp> & __H1psum, std::complex<_Tp> & __H2sum, std::complex<_Tp> & __H2psum)`

Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to `nterms` (less than 5) to achieve relative error `eps`.

Parameters

in	<code>__p</code>	
in	<code>__p2</code>	
in	<code>__num2</code>	
in	<code>__zetam3hf</code>	
in	<code>__Aip</code>	The Airy function value $Ai()$.
in	<code>__o4dp</code>	
in	<code>__Aim</code>	The Airy function value $Ai()$.
in	<code>__o4dm</code>	
in	<code>__od2p</code>	
in	<code>__od0dp</code>	
in	<code>__od2m</code>	
in	<code>__od0dm</code>	
in	<code>__eps</code>	The error tolerance
out	<code>__H1sum</code>	The Hankel function of the first kind.
out	<code>__H1psum</code>	The derivative of the Hankel function of the first kind.
out	<code>__H2sum</code>	The Hankel function of the second kind.
out	<code>__H2psum</code>	The derivative of the Hankel function of the second kind.

Definition at line 325 of file sf_hankel.tcc.

Referenced by `__hankel_uniform_olver()`.

9.3.1.154 `template<typename _Tp> _Tp std::__detail::__harmonic_number (unsigned int __n)`

Definition at line 3248 of file sf_gamma.tcc.

References `std::__detail::__Factorial_table< _Tp >::__n`, `_S_harmonic_denom`, `_S_harmonic_numer`, and `_S_num_↵harmonic_numer`.

9.3.1.155 `template<typename _Tp> std::vector<__gnu_cxx::__quadrature_point_t<_Tp>> >
std::__detail::__hermite_zeros (unsigned int __n, _Tp __proto = _Tp{ })`

Build a vector of the Gauss-Hermite integration rule abscissae and weights.

Definition at line 246 of file sf_hermite.tcc.

9.3.1.156 `template<typename _Tp> _Tp std::__detail::__heuman_lambda (_Tp __k, _Tp __phi)`

Return the Heuman lambda function.

Definition at line 986 of file sf_ellint.tcc.

References `__comp_ellint_1()`, `__ellint_1()`, `__ellint_rf()`, `__ellint_rj()`, and `__jacobi_zeta()`.

9.3.1.157 `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 773 of file sf_zeta.tcc.

References `__hurwitz_zeta_euler_maclaurin()`, and `__riemann_zeta()`.

Referenced by `__psi()`.

9.3.1.158 `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"0160tas

Parameters

<code>__s</code>	The argument $s \neq 1$
<code>__a</code>	The scale parameter $a > -1$

Definition at line 725 of file `sf_zeta.tcc`.

References `_S_Euler_Maclaurin_zeta`.

Referenced by `__hurwitz_zeta()`.

9.3.1.159 `template<typename _Tp> std::complex<_Tp> std::__detail::__hurwitz_zeta_polylog (_Tp __s, std::complex<_Tp> __a)`

Return the Hurwitz Zeta function for real s and complex a . This uses Jonquiere's identity:

$$\frac{(i2\pi)^s}{\Gamma(s)} \zeta(a, 1-s) = Li_s(e^{i2\pi a}) + (-1)^s Li_s(e^{-i2\pi a})$$

Parameters

<code>__s</code>	The real argument
<code>__a</code>	The complex parameter

Todo This `__hurwitz_zeta_polylog` prefactor is prone to overflow. positive integer orders s ?

Definition at line 1050 of file `sf_polylog.tcc`.

References `__gamma()`, and `__polylog_exp()`.

9.3.1.160 `template<typename _Tp> std::complex<_Tp> std::__detail::__hydrogen (unsigned int __n, unsigned int __l, unsigned int __m, _Tp __Z, _Tp __r, _Tp __theta, _Tp __phi)`

Return the bound-state Coulomb wave-function.

Definition at line 49 of file sf_hydrogen.tcc.

References `__assoc_laguerre()`, `__log_gamma()`, `__psi()`, and `__sph_legendre()`.

9.3.1.161 `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 814 of file sf_hyperg.tcc.

References `__hyperg_luke()`, `__hyperg_reflect()`, `__hyperg_series()`, `__log_gamma()`, and `__log_gamma_sign()`.

9.3.1.162 `template<typename _Tp> _Tp std::__detail::__hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.

Definition at line 405 of file sf_hyperg.tcc.

Referenced by `__hyperg()`.

9.3.1.163 `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:

$${}_2F_1(a, b; a+b+m; x) = \frac{\Gamma(m)\Gamma(a+b+m)}{\Gamma(a+m)\Gamma(b+m)} \sum_{k=0}^{m-1} \frac{(m+a)_k(m+b)_k}{k!(1-m)_k} (1-x)^k + (-1)^m$$

Definition at line 540 of file `sf_hyperg.tcc`.

References `__hyperg_series()`, `__log_gamma()`, `__log_gamma_sign()`, and `__psi()`.

Referenced by `__hyperg()`.

9.3.1.164 `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 374 of file sf_hyperg.tcc.

Referenced by `__hyperg()`, and `__hyperg_reflect()`.

9.3.1.165 `template<typename _Tp> _Tp std::__detail::__ibeta_cont_frac (_Tp __a, _Tp __b, _Tp __x)`

Return the regularized incomplete beta function, $I_x(a, b)$, of arguments *a*, *b*, and *x*.

Parameters

<code>__a</code>	The first parameter
<code>__b</code>	The second parameter
<code>__x</code>	The argument

Definition at line 239 of file sf_beta.tcc.

Referenced by `__beta_inc()`.

9.3.1.166 `template<typename _Tp> __gnu_cxx::__jacobi_t<_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 447 of file sf_theta.tcc.

9.3.1.167 `template<typename _Tp> _Tp std::__detail::__jacobi_zeta (_Tp __k, _Tp __phi)`

Return the Jacobi zeta function.

Definition at line 949 of file sf_ellint.tcc.

References `__comp_ellint_1()`, and `__ellint_rj()`.

Referenced by `__heuman_lambda()`.

9.3.1.168 `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

\leftrightarrow _n	The order of the Laguerre polynomial.
\leftrightarrow _x	The argument of the Laguerre polynomial.

Returns

The value of the Laguerre polynomial of order n and argument x.

Definition at line 384 of file sf_laguerre.tcc.

9.3.1.169 `template<typename _Tp> std::vector<__gnu_cxx::__quadrature_point_t<_Tp>> std::__detail::__laguerre_zeros (unsigned int _n, _Tp __alpha)`

Return an array of abscissae and weights for the Gauss-Laguerre rule.

Definition at line 223 of file sf_laguerre.tcc.

References `__gnu_cxx::lgamma()`.

9.3.1.170 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_binet1p (_Tp __z)`

Return the Binet function $J(1+z)$ by the Lanczos method. The Binet function is the log of the scaled Gamma function $\log(\Gamma^*(z))$ defined by

$$J(z) = \log(\Gamma^*(z)) = \log(\Gamma(z)) + z - \left(z - \frac{1}{2}\right) \log(z) - \log(2\pi)$$

or

$$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

where $\Gamma(z)$ is the gamma function.

Parameters

\leftrightarrow _z	The argument of the log of the gamma function.
-------------------------	--

Returns

The logarithm of the gamma function.

Definition at line 2102 of file sf_gamma.tcc.

References `std::__detail::__Factorial_table<_Tp>::__n`.

Referenced by `__lanczos_log_gamma1p()`.

9.3.1.171 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_log_gamma1p (_Tp __z)`

Return the logarithm of the gamma function $\log(\Gamma(1 + z))$ by the Lanczos method.

If the argument is real, the log of the absolute value of the Gamma function is returned. The sign to be applied to the exponential of this log Gamma can be recovered with a call to `__log_gamma_sign`.

For complex argument the fully complex log of the gamma function is returned.

Parameters

<code>__z</code>	The argument of the log of the gamma function.
------------------	--

Returns

The logarithm of the gamma function.

Definition at line 2136 of file `sf_gamma.tcc`.

References `__lanczos_binet1p()`, and `__sin_pi()`.

9.3.1.172 `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 130 of file `sf_legendre.tcc`.

9.3.1.173 `template<typename _Tp> std::vector<__gnu_cxx::__quadrature_point_t<_Tp>> std::__detail::__legendre_zeros (unsigned int __l, _Tp proto = _Tp{})`

Build a list of zeros and weights for the Gauss-Legendre integration rule for the Legendre polynomial of degree l .

Definition at line 372 of file `sf_legendre.tcc`.

9.3.1.174 `template<typename _Tp> _Tp std::__detail::__log_binomial (unsigned int __n, unsigned int __k)`

Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

.

Parameters

\leftrightarrow __n	The first argument of the binomial coefficient.
\leftrightarrow __k	The second argument of the binomial coefficient.

Returns

The logarithm of the binomial coefficient.

Definition at line 2411 of file sf_gamma.tcc.

References `__log_gamma()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__binomial()`.

9.3.1.175 `template<typename _Tp> _Tp std::__detail::__log_binomial (_Tp __nu, unsigned int __k)`

Return the logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

.

Parameters

__nu	The first argument of the binomial coefficient.
__k	The second argument of the binomial coefficient.

Returns

The logarithm of the binomial coefficient.

Definition at line 2448 of file sf_gamma.tcc.

References `__log_gamma()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

9.3.1.176 `template<typename _Tp> _Tp std::__detail::__log_binomial_sign (_Tp __nu, unsigned int __k)`

Return the sign of the exponentiated logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu + 1)}{\Gamma(\nu - k + 1)\Gamma(k + 1)}$$

The binomial coefficients are generated by:

$$(1 + t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

Parameters

<code>__nu</code>	The first argument of the binomial coefficient.
<code>__k</code>	The second argument of the binomial coefficient.

Returns

The sign of the gamma function.

Definition at line 2479 of file sf_gamma.tcc.

References `__log_gamma_sign()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__binomial()`.

9.3.1.177 `template<typename _Tp> std::complex<_Tp> std::__detail::__log_binomial_sign (std::complex<_Tp> __nu, unsigned int __k)`

Definition at line 2494 of file sf_gamma.tcc.

9.3.1.178 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (_Tp __x)`

Definition at line 1643 of file sf_gamma.tcc.

References `__cos_pi()`, and `__log_gamma()`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

9.3.1.179 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (int __n)`

Return the logarithm of the double factorial of the integer n.

The double factorial is defined for integral n by:

$$n!! = 135...(n-2)n, \text{ odd } n!! = 246...(n-2)n, \text{ even } -1!! = 10!! = 1$$

The double factorial is defined for odd negative integers in the obvious way:

$$(-2m-1)!! = 1/(1(-1)(-3)...(-2m+1)(-2m-1)) = \frac{(-1)^m}{(2m-1)!!}$$

for $n = -2m - 1$.

Definition at line 1709 of file `sf_gamma.tcc`.

References `__log_double_factorial()`, `std::__detail::_Factorial_table<_Tp>::__log_factorial`, `std::__detail::_Factorial_table<_Tp>::__n`, `_S_double_factorial_table`, and `_S_neg_double_factorial_table`.

9.3.1.180 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_factorial (unsigned int __n)`

Return the logarithm of the factorial of the integer n.

The factorial is:

$$n! = 12...(n-1)n, 0! = 1$$

Definition at line 1633 of file `sf_gamma.tcc`.

References `__log_gamma()`, `std::__detail::_Factorial_table<_Tp>::__n`, and `_S_factorial_table`.

9.3.1.181 `template<typename _Tp > _Tp std::__detail::__log_falling_factorial (_Tp __a, _Tp __nu)`

Return the logarithm of the falling factorial function or the lower Pochhammer symbol. The lower Pochhammer symbol is defined by

$$a^{\underline{n}} = \Gamma(a+1)/\Gamma(a-\nu+1) = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1$$

In particular, $n^{\underline{n}} = n!$. Thus this function returns

$$\ln[a^{\underline{n}}] = \ln[\Gamma(a+1)] - \ln[\Gamma(a-\nu+1)], \ln[a^{\underline{0}}] = 0$$

Many notations exist for this function:

$$(a)_{\nu}$$

,

$$\left\{ \begin{matrix} a \\ \nu \end{matrix} \right\}$$

, and others.

Definition at line 3012 of file `sf_gamma.tcc`.

References `__falling_factorial()`, `__gnu_cxx::__fp_is_integer()`, and `__log_gamma()`.

9.3.1.182 `template<typename _Tp > _Tp std::__detail::__log_gamma (_Tp __a)`

Return $\log(|\Gamma(a)|)$. This will return values even for $a < 0$. To recover the sign of $\Gamma(a)$ for any argument use `__log_gamma_sign`.

Parameters

\leftrightarrow _a	The argument of the log of the gamma function.
-------------------------	--

Returns

The logarithm of the gamma function.

Definition at line 2302 of file sf_gamma.tcc.

References `__sin_pi()`, and `__spouge_log_gamma1p()`.

Referenced by `__beta_inc()`, `__beta_lgamma()`, `__cyl_bessel_ij_series()`, `__falling_factorial()`, `__gamma()`, `__gamma_cont_frac()`, `__gamma_series()`, `__hydrogen()`, `__hyperg()`, `__hyperg_reflect()`, `__log_binomial()`, `__log_double_factorial()`, `__log_factorial()`, `__log_falling_factorial()`, `__log_gamma()`, `__log_rising_factorial()`, `__poly_laguerre_large_n()`, `__polylog_exp_neg()`, `__polylog_exp_pos()`, `__psi()`, `__riemann_zeta()`, `__rising_factorial()`, and `__sph_legendre()`.

9.3.1.183 `template<typename _Tp> std::complex<_Tp> std::__detail::__log_gamma (std::complex<_Tp> __a)`

Return $\log(\Gamma(a))$ for complex argument.

Parameters

\leftrightarrow _a	The complex argument of the log of the gamma function.
-------------------------	--

Returns

The complex logarithm of the gamma function.

Definition at line 2337 of file sf_gamma.tcc.

References `__gnu_cxx::__fp_is_integer()`, `std::__detail::_Factorial_table<_Tp>::__log_factorial`, `__log_gamma()`, `std::__detail::_Factorial_table<_Tp>::__n`, `__sin_pi()`, `__spouge_log_gamma1p()`, and `_S_factorial_table`.

9.3.1.184 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (_Tp __x)`

Return $\log(\Gamma(x))$ by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.

Parameters

\leftrightarrow _x	The argument of the log of the gamma function.
-------------------------	--

Returns

The logarithm of the gamma function.

Definition at line 1736 of file sf_gamma.tcc.

9.3.1.185 `template<typename _Tp> _Tp std::__detail::__log_gamma_sign (_Tp __a)`

Return the sign of $\Gamma(x)$. At nonpositive integers zero is returned indicating $\Gamma(x)$ is undefined.

Parameters

<code>__a</code>	The argument of the gamma function.
------------------	-------------------------------------

Returns

The sign of the gamma function.

Definition at line 2378 of file sf_gamma.tcc.

Referenced by `__beta_inc()`, `__beta_lgamma()`, `__falling_factorial()`, `__gamma()`, `__gamma_cont_frac()`, `__gamma_↔series()`, `__hyperg()`, `__hyperg_reflect()`, `__log_binomial_sign()`, and `__rising_factorial()`.

9.3.1.186 `template<typename _Tp> std::complex<_Tp> std::__detail::__log_gamma_sign (std::complex<_Tp> __a)`

Definition at line 2390 of file sf_gamma.tcc.

9.3.1.187 `template<typename _Tp> _Tp std::__detail::__log_rising_factorial (_Tp __a, _Tp __nu)`

Return the logarithm of the rising factorial function or the (upper) Pochhammer symbol. The Pochhammer symbol is defined for integer order by

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(a) = \prod_{k=0}^{\nu-1} (a + k), (a)_0 = 1$$

Thus this function returns

$$\ln[a^{\overline{\nu}}] = \ln[\Gamma(a + \nu)] - \ln[\Gamma(a)], \ln[(a)_0] = 0$$

Many notations exist for this function:

$$(a)_{\nu}$$

(especially in the literature of special functions),

$$\left[\begin{matrix} a \\ \nu \end{matrix} \right]$$

, and others.

Definition at line 3161 of file sf_gamma.tcc.

References `__log_gamma()`, and `__rising_factorial()`.

9.3.1.188 `template<typename _Tp> _Tp std::__detail::__log_stirling_1 (unsigned int __n, unsigned int __m)`

Return the logarithm of the absolute value of Stirling number of the first kind.

Definition at line 318 of file `sf_stirling.tcc`.

9.3.1.189 `template<typename _Tp> _Tp std::__detail::__log_stirling_1_sign (unsigned int __n, unsigned int __m)`
`[inline]`

Return the sign of the exponent of the logarithm of the Stirling number of the first kind.

Definition at line 336 of file `sf_stirling.tcc`.

9.3.1.190 `template<typename _Tp> _Tp std::__detail::__log_stirling_2 (unsigned int __n, unsigned int __m)`

Return the Stirling number of the second kind.

Todo Look into asymptotic solutions.

Definition at line 178 of file `sf_stirling.tcc`.

9.3.1.191 `template<typename _Tp> _Tp std::__detail::__logint (const _Tp __x)`

Return the logarithmic integral $li(x)$.

The logarithmic integral is given by

$$li(x) = Ei(\log(x))$$

Parameters

<code>__x</code>	The argument of the logarithmic integral function.
------------------	--

Returns

The logarithmic integral.

Definition at line 538 of file `sf_expint.tcc`.

References `__expint()`.

9.3.1.192 `template<typename _Tp> _Tp std::__detail::__logistic_cdf (_Tp __a, _Tp __b, _Tp __x)`

Return the logistic cumulative distribution function.

The formula for the logistic probability function is

$$cdf(x|a, b) = \frac{e^{(x-a)/b}}{1 + e^{(x-a)/b}}$$

where $b > 0$.

Definition at line 688 of file `sf_distributions.tcc`.

9.3.1.193 `template<typename _Tp> _Tp std::__detail::__logistic_pdf (_Tp __a, _Tp __b, _Tp __x)`

Return the logistic probability density function.

The formula for the logistic probability density function is

$$p(x|a, b) = \frac{e^{(x-a)/b}}{b[1 + e^{(x-a)/b}]^2}$$

where $b > 0$.

Definition at line 670 of file `sf_distributions.tcc`.

9.3.1.194 `template<typename _Tp> _Tp std::__detail::__lognormal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)`

Return the lognormal cumulative probability density function.

The formula for the lognormal cumulative probability density function is

$$F(x|\mu, \sigma) = \frac{1}{2} \left[1 - \operatorname{erf} \left(\frac{\ln x - \mu}{\sqrt{2}\sigma} \right) \right]$$

Definition at line 287 of file `sf_distributions.tcc`.

9.3.1.195 `template<typename _Tp> _Tp std::__detail::__lognormal_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`

Return the lognormal probability density function.

The formula for the lognormal probability density function is

$$f(x|\mu, \sigma) = \frac{e^{(\ln x - \mu)^2 / 2\sigma^2}}{\sigma\sqrt{2\pi}}$$

Definition at line 259 of file `sf_distributions.tcc`.

9.3.1.196 `template<typename _Tp> _Tp std::__detail::__normal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)`

Return the normal cumulative probability density function.

The formula for the normal cumulative probability density function is

$$F(x|\mu, \sigma) = \frac{1}{2} \left[1 - \operatorname{erf}\left(\frac{x - \mu}{\sqrt{2}\sigma}\right) \right]$$

Definition at line 238 of file `sf_distributions.tcc`.

9.3.1.197 `template<typename _Tp> _Tp std::__detail::__normal_pdf (_Tp __mu, _Tp __sigma, _Tp __x)`

Return the normal probability density function.

The formula for the normal probability density function is

$$f(x|\mu, \sigma) = \frac{e^{(x-\mu)^2/2\sigma^2}}{\sigma\sqrt{2\pi}}$$

Definition at line 210 of file `sf_distributions.tcc`.

9.3.1.198 `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()`.

9.3.1.199 `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 2767 of file sf_gamma.tcc.

References `__gnu_cxx::__fp_is_integer()`, `__gamma_cont_frac()`, and `__gamma_series()`.

Referenced by `__chi_squared_pdf()`.

9.3.1.200 `template<typename _Tp> std::complex<_Tp> std::__detail::__polar_pi (_Tp __rho, _Tp __phi_pi) [inline]`

Reperiodized complex constructor.

Definition at line 397 of file sf_trig.tcc.

References `__gnu_cxx::__sincos_t<_Tp>::__cos_v`, `__gnu_cxx::__sincos_t<_Tp>::__sin_v`, and `__sincos_pi()`.

Referenced by `__cyl_bessel_jn_neg_arg()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__polylog_exp_neg()`, and `__polylog_exp_pos()`.

9.3.1.201 `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:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

An explicit series formula is:

$$H_n(x) = \sum_{k=0}^m \frac{(-1)^k}{k!(n-2k)!} (2x)^{n-2k} \text{ where } m = \left\lfloor \frac{n}{2} \right\rfloor$$

The Hermite polynomial obeys a reflection formula:

$$H_n(-x) = (-1)^n H_n(x)$$

Parameters

\leftrightarrow _n	The order of the Hermite polynomial.
\leftrightarrow _x	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x.

Definition at line 184 of file sf_hermite.tcc.

References `__poly_hermite_asymp()`, and `__poly_hermite_recursion()`.

9.3.1.202 `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 also

"Asymptotic analysis of the Hermite polynomials from their differential-difference equation", Diego Dominici, [arXiv:math/0601078v1 \[math.CA\]](#) 4 Jan 2006

Parameters

\leftrightarrow _n	The order of the Hermite polynomial.
\leftrightarrow _x	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x.

Definition at line 115 of file sf_hermite.tcc.

References `__airy()`.

Referenced by `__poly_hermite()`.

9.3.1.203 `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

\longleftrightarrow __n	The order of the Hermite polynomial.
\longleftrightarrow __x	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x.

Definition at line 71 of file sf_hermite.tcc.

Referenced by `__poly_hermite()`.

9.3.1.204 `template<typename _Tp> _Tp std::__detail::__poly_jacobi (unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)`

Compute the Jacobi polynomial by recursion on n :

$$2n(\alpha+\beta+n)(\alpha+\beta+2n-2)P_n^{(\alpha,\beta)}(x) = (\alpha+\beta+2n-1)((\alpha^2-\beta^2)+x(\alpha+\beta+2n-2)(\alpha+\beta+2n))P_{n-1}^{(\alpha,\beta)}(x) - 2(\alpha+n-1)(\beta+n-1)(\alpha+\beta+2n-2)P_{n-2}^{(\alpha,\beta)}(x)$$

Definition at line 59 of file sf_jacobi.tcc.

References `__beta()`.

Referenced by `__poly_radial_jacobi()`.

9.3.1.205 `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 314 of file `sf_laguerre.tcc`.

References `__poly_laguerre_hyperg()`, `__poly_laguerre_large_n()`, and `__poly_laguerre_recursion()`.

9.3.1.206 `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 132 of file sf_laguerre.tcc.

Referenced by __poly_laguerre().

9.3.1.207 `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 75 of file sf_laguerre.tcc.

References __log_gamma(), and __sin_pi().

Referenced by __poly_laguerre().

9.3.1.208 `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 190 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

9.3.1.209 `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:

$$P_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} (x^2 - 1)^l$$

This can be expressed as a series:

$$P_l(x) = \frac{1}{2^l l!} \sum_{k=0}^{\lfloor l/2 \rfloor} \frac{(-1)^k (2l - 2k)!}{k! (l - k)! (l - 2k)!} x^{l-2k}$$

Parameters

\leftrightarrow _l	The order of the Legendre polynomial. $l \geq 0$.
\leftrightarrow _x	The argument of the Legendre polynomial.

Definition at line 82 of file sf_legendre.tcc.

Referenced by __assoc_legendre_p(), and __sph_legendre().

9.3.1.210 `template<typename _Tp> _Tp std::__detail::__poly_prob_hermite_recursion (unsigned int __n, _Tp __x)`

This routine returns the Probabilists Hermite polynomial of order n: $He_n(x)$ by recursion on n.

The Hermite polynomial is defined by:

$$He_n(x) = (-1)^n e^{x^2/2} \frac{d^n}{dx^n} e^{-x^2/2}$$

or

$$He_n(x) = \frac{1}{2^{-n/2}} H_n \left(\frac{x}{\sqrt{2}} \right)$$

where $H_n(x)$ is the Physicists Hermite function.

Parameters

\leftrightarrow _n	The order of the Hermite polynomial.
\leftrightarrow _x	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x.

Definition at line 217 of file sf_hermite.tcc.

9.3.1.211 `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

$$R_n^m(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k! \left(\frac{n+m}{2} - k\right)! \left(\frac{n-m}{2} - k\right)!} \rho^{n-2k}$$

for $n - m$ even and identically 0 for $n - m$ odd. The radial polynomials can be related to the Zernike polynomials:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative m, n .

See also

zernike for details on the Zernike polynomials.

Principals of Optics, 7th edition, Max Born and Emil Wolf, Cambridge University Press, 1999, pp 523-525 and 905-910.

Template Parameters

<code>_Tp</code>	The real type of the radial coordinate
------------------	--

Parameters

<code>__n</code>	The non-negative degree.
<code>__m</code>	The non-negative azimuthal order
<code>__rho</code>	The radial argument

Definition at line 144 of file `sf_jacobi.tcc`.

References `__poly_jacobi()`.

Referenced by `__zernike()`, `__gnu_cxx::radpolyf()`, and `__gnu_cxx::radpolyl()`.

9.3.1.212 `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 987 of file `sf_polylog.tcc`.

References `__gnu_cxx::__fp_is_equal()`, `__gnu_cxx::__fp_is_integer()`, `__gnu_cxx::__fp_is_zero()`, and `__polylog_exp()`.

Referenced by `__dirichlet_beta()`, `__dirichlet_eta()`, and `__polylog()`.

9.3.1.213 `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 1028 of file `sf_polylog.tcc`.

References `__polylog()`, and `__polylog_exp()`.

9.3.1.214 `template<typename _Tp, typename _ArgType> __gnu_cxx::__promote_fp_t<std::complex<_Tp>, _ArgType> std::__detail::__polylog_exp (_Tp __s, _ArgType __w)`

This is the frontend function which calculates $Li_s(e^w)$. First we branch into different parts depending on the properties of s . This function is the same irrespective of a real or complex w , hence the template parameter `ArgType`.

Note

: I *really* wish we could return a `variant<Tp, std::complex<Tp>>`.

Parameters

<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 951 of file `sf_polylog.tcc`.

References `__gnu_cxx::__fp_is_integer()`, `__polylog_exp_neg_int()`, `__polylog_exp_neg_real()`, `__polylog_exp_pos_↵
int()`, `__polylog_exp_pos_real()`, and `__polylog_exp_sum()`.

Referenced by `__bose_einstein()`, `__clausen()`, `__clausen_cl()`, `__clausen_sl()`, `__fermi_dirac()`, `__hurwitz_zeta_↵
polylog()`, and `__polylog()`.

9.3.1.215 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_asyp (_Tp __s, std::complex<_Tp> ↵
__w)`

This function implements the asymptotic series for the polylog. It is given by

$$2 \sum_{k=0}^{\infty} \zeta(2k) w^{s-2k} / \Gamma(s-2k+1) - i\pi w^{s-1} / \Gamma(s)$$

for $Re(w) \gg 1$

Don't check this against Mathematica 8. For real w the imaginary part of the polylog is given by $Im(Li_s(e^w)) = -\pi w^{s-1} / \Gamma(s)$. Check this relation for any benchmark that you use.

Parameters

<code>↵ __s</code>	the real index s .
<code>↵ __w</code>	the large complex argument w .

Returns

the value of the polylogarithm.

Definition at line 564 of file `sf_polylog.tcc`.

References `__gamma_reciprocal()`.

Referenced by `__polylog_exp_neg_int()`, `__polylog_exp_neg_real()`, `__polylog_exp_pos_int()`, and `__polylog_exp_↵
pos_real()`.

9.3.1.216 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg (_Tp __s, std::complex<_Tp> ↵
__w)`

This function treats the cases of negative real index s . Theoretical convergence is present for $|w| < 2\pi$. We use an optimized version of

$$Li_s(e^w) = \Gamma(1-s)(-w)^{s-1} + \frac{(2\pi)^{-s}}{\pi} A_p(w)$$

$$A_p(w) = \sum_k \frac{\Gamma(1+k-s)}{k!} \sin\left(\frac{\pi}{2}(s-k)\right) \left(\frac{w}{2\pi}\right)^k \zeta(1+k-s)$$

Parameters

\leftarrow _s	The negative real index
\leftarrow _w	The complex argument

Returns

The value of the polylogarithm.

Definition at line 328 of file sf_polylog.tcc.

References `__log_gamma()`, `__polar_pi()`, and `__riemann_zeta_m_1()`.

Referenced by `__polylog_exp_neg_int()`, and `__polylog_exp_neg_real()`.

9.3.1.217 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg (int __n, std::complex<_Tp> __w)`

Compute the polylogarithm for negative integer order.

$$Li_{-p}(e^w) = p!(-w)^{-(p+1)} - \sum_{k=0}^{\infty} \frac{B_{p+2k+q+1}}{(p+2k+q+1)!} \frac{(p+2k+q)!}{(2k+q)!} w^{2k+q}$$

where $q = (p+1) \bmod 2$.

Parameters

\leftarrow _n	the negative integer index $n = -p$.
\leftarrow _w	the argument w.

Returns

the value of the polylogarithm.

Definition at line 414 of file sf_polylog.tcc.

References `__gnu_cxx::__fp_is_equal()`, `__gnu_cxx::__fp_is_zero()`, `_Num_Euler_Maclaurin_zeta`, and `_S_Euler_` \leftarrow Maclaurin_zeta.

9.3.1.218 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg_int (int __s, std::complex<_Tp> __w)`

This treats the case where s is a negative integer.

Parameters

$_s$	a negative integer.
$_w$	an arbitrary complex number

Returns

the value of the polylogarithm.

Definition at line 746 of file sf_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__gnu_cxx::__fp_is_equal()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_sum()`.

Referenced by `__polylog_exp()`.

9.3.1.219 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg_int (int __s, _Tp __w)`

This treats the case where s is a negative integer and w is a real.

Parameters

$_s$	a negative integer.
$_w$	the argument.

Returns

the value of the polylogarithm.

Definition at line 790 of file sf_polylog.tcc.

References `__gnu_cxx::__fp_is_zero()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_sum()`.

9.3.1.220 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg_real (_Tp __s, std::complex<_Tp> __w)`

Return the polylog where s is a negative real value and for complex argument. Now we branch depending on the properties of w in the specific functions

Parameters

$_s$	A negative real value that does not reduce to a negative integer.
$_w$	The complex argument.

Returns

The value of the polylogarithm.

Definition at line 891 of file sf_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_sum()`.

Referenced by `__polylog_exp()`.

9.3.1.221 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg_real (_Tp __s, _Tp __w)`

Return the polylog where s is a negative real value and for real argument. Now we branch depending on the properties of w in the specific functions.

Parameters

$_s$	A negative real value.
$_w$	A real argument.

Returns

The value of the polylogarithm.

Definition at line 922 of file sf_polylog.tcc.

References `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_sum()`.

9.3.1.222 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos (unsigned int __s, std::complex<_Tp> __w)`

This function treats the cases of positive integer index s for complex argument w.

$$Li_s(e^w) = \sum_{k=0, k! \neq s-1} \zeta(s-k) \frac{w^k}{k!} + [H_{s-1} - \log(-w)] \frac{w^{s-1}}{(s-1)!}$$

The radius of convergence is $|w| < 2\pi$. Note that this series involves a $\log(-x)$. gcc and Mathematica differ in their implementation of $\log(e^{i\pi})$: gcc: $\log(e^{+i\pi}) = + - i\pi$ whereas Mathematica doesn't preserve the sign in this case: $\log(e^{+i\pi}) = +i\pi$

Parameters

\leftrightarrow _s	the positive integer index.
\leftrightarrow _w	the argument.

Returns

the value of the polylogarithm.

Definition at line 180 of file sf_polylog.tcc.

References `__riemann_zeta()`.

Referenced by `__polylog_exp_pos_int()`, and `__polylog_exp_pos_real()`.

9.3.1.223 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos (unsigned int __s, _Tp __w)`

This function treats the cases of positive integer index s for real argument w .

This specialization is worthwhile to catch the differing behaviour of $\log(x)$.

$$Li_s(e^w) = \sum_{k=0, k! \neq s-1} \zeta(s-k) \frac{w^k}{k!} + [H_{s-1} - \log(-w)] \frac{w^{s-1}}{(s-1)!}$$

The radius of convergence is $|w| < 2\pi$. Note that this series involves a $\log(-x)$. gcc and Mathematica differ in their implementation of $\log(e^{i\pi})$: gcc: $\log(e^{+ - i\pi}) = + - i\pi$ whereas Mathematica doesn't preserve the sign in this case: $\log(e^{+ - i\pi}) = + i\pi$

Parameters

\leftrightarrow _s	the positive integer index.
\leftrightarrow _w	the argument.

Returns

the value of the polylogarithm.

Definition at line 256 of file sf_polylog.tcc.

References `__riemann_zeta()`.

9.3.1.224 `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 477 of file sf_polylog.tcc.

References `__gamma()`, `__log_gamma()`, `__polar_pi()`, and `__riemann_zeta()`.

9.3.1.225 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos_int (unsigned int __s, std::complex<_Tp> __w)`

Here s is a positive integer and the function descends into the different kernels depending on w.

Parameters

<code>__s</code>	a positive integer.
<code>__w</code>	an arbitrary complex number.

Returns

The value of the polylogarithm.

Definition at line 639 of file sf_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__gnu_cxx::__fp_is_equal()`, `__gnu_cxx::__fp_is_zero()`, `__polylog_↵exp_asymp()`, `__polylog_exp_pos()`, and `__polylog_exp_sum()`.

Referenced by `__polylog_exp()`.

9.3.1.226 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos_int (unsigned int __s, _Tp __w)`

Here `s` is a positive integer and the function descends into the different kernels depending on `w`.

Parameters

<code>↵ __s</code>	a positive integer
<code>↵ __w</code>	an arbitrary real argument <code>w</code>

Returns

the value of the polylogarithm.

Definition at line 698 of file `sf_polylog.tcc`.

References `__gnu_cxx::__fp_is_zero()`, `__polylog_exp_asymp()`, `__polylog_exp_pos()`, and `__polylog_exp_sum()`.

9.3.1.227 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos_real (_Tp __s, std::complex<_Tp> __w)`

Return the polylog where `s` is a positive real value and for complex argument.

Parameters

<code>↵ __s</code>	A positive real number.
<code>↵ __w</code>	the complex argument.

Returns

The value of the polylogarithm.

Definition at line 817 of file `sf_polylog.tcc`.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__gnu_cxx::__fp_is_equal()`, `__gnu_cxx::__fp_is_zero()`, `__polylog_↵exp_asymp()`, `__polylog_exp_pos()`, `__polylog_exp_sum()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp()`.

9.3.1.228 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos_real (_Tp __s, _Tp __w)`

Return the polylog where s is a positive real value and the argument is real.

Parameters

<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 857 of file sf_polylog.tcc.

References `__gnu_cxx::__fp_is_equal()`, `__gnu_cxx::__fp_is_zero()`, `__polylog_exp_asymp()`, `__polylog_exp_pos()`, `__polylog_exp_sum()`, and `__riemann_zeta()`.

9.3.1.229 `template<typename _PowTp, typename _Tp> _Tp std::__detail::__polylog_exp_sum (_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:

$$Li_s(e^z) = \sum_{k=1}^{\infty} e^{kz} k^{-s}$$

Parameters

<code>__s</code>	is an arbitrary type, integral or float.
<code>__w</code>	something with a negative real part.

Returns

the value of the polylogarithm.

Definition at line 608 of file sf_polylog.tcc.

Referenced by `__polylog_exp()`, `__polylog_exp_neg_int()`, `__polylog_exp_neg_real()`, `__polylog_exp_pos_int()`, and `__polylog_exp_pos_real()`.

9.3.1.230 `template<typename _Tp> _Tp std::__detail::__psi (unsigned int __n)`

Return the digamma function of integral argument. The digamma or $\psi(x)$ function is defined as the logarithmic derivative of the gamma function:

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

The digamma series for integral argument is given by:

$$\psi(n) = -\gamma_E + \sum_{k=1}^{n-1} \frac{1}{k}$$

The latter sum is called the harmonic number, H_n .

Definition at line 3279 of file `sf_gamma.tcc`.

Referenced by `__expint_En_series()`, `__hydrogen()`, `__hyperg_reflect()`, and `__psi()`.

9.3.1.231 `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 3365 of file `sf_gamma.tcc`.

References `__gnu_cxx::__fp_is_half_odd_integer()`, `__gnu_cxx::__fp_is_integer()`, `std::__detail::_Factorial_table<_Tp>::__n`, `__psi()`, `__psi_asymp()`, and `__tan_pi()`.

9.3.1.232 `template<typename _Tp> _Tp std::__detail::__psi (unsigned int __n, _Tp __x)`

Return the polygamma function $\psi^{(n)}(x)$.

The polygamma function is related to the Hurwitz zeta function:

$$\psi^{(n)}(x) = (-1)^{n+1} n! \zeta(n+1, x)$$

Definition at line 3421 of file `sf_gamma.tcc`.

References `__hurwitz_zeta()`, `__log_gamma()`, and `__psi()`.

9.3.1.233 `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 3334 of file `sf_gamma.tcc`.

Referenced by `__psi()`.

9.3.1.234 `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 3303 of file `sf_gamma.tcc`.

9.3.1.235 `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 2801 of file `sf_gamma.tcc`.

References `__gnu_cxx::__fp_is_integer()`, `__gamma_cont_frac()`, and `__gamma_series()`.

Referenced by `__chi_squared_pdfc()`.

9.3.1.236 `template<typename _Tp> _Tp std::__detail::__rice_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`

Return the Rice probability density function.

The formula for the Rice probability density function is

$$p(x|\nu, \sigma) = \frac{x}{\sigma^2} \exp\left(-\frac{x^2 + \nu^2}{2\sigma^2}\right) I_0\left(\frac{x\nu}{\sigma^2}\right)$$

where $I_0(x)$ is the modified Bessel function of the first kind of order 0 and $\nu \geq 0$ and $\sigma > 0$.

Definition at line 186 of file `sf_distributions.tcc`.

References `__cyl_bessel_i()`.

9.3.1.237 `template<typename _Tp> _Tp std::__detail::__riemann_zeta (_Tp __s)`

Return the Riemann zeta function $\zeta(s)$.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} k^{-s} \text{ for } \Re(s) > 1 \quad \frac{(2\pi)^s}{\pi} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s) \zeta(1-s) \text{ for } \Re(s) < 1$$

Parameters

<code>__s</code>	The argument
------------------	--------------

Todo Global double sum or MacLaurin series in `riemann_zeta`?

Definition at line 663 of file `sf_zeta.tcc`.

References `__gnu_cxx::__fp_is_integer()`, `__gamma()`, `__log_gamma()`, `__riemann_zeta_glob()`, `__riemann_zeta_m_1()`, `__riemann_zeta_product()`, `__riemann_zeta_sum()`, and `__sin_pi()`.

Referenced by `__dirichlet_lambda()`, `__hurwitz_zeta()`, `__polylog_exp_pos()`, and `__polylog_exp_pos_real()`.

9.3.1.238 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_euler_maclaurin (_Tp __s)`

Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.

This is a specialization of the code for the Hurwitz zeta function.

Definition at line 300 of file `sf_zeta.tcc`.

References `_S_Euler_Maclaurin_zeta`.

9.3.1.239 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_glob (_Tp __s)`

Definition at line 410 of file sf_zeta.tcc.

References `__gnu_cxx::__fp_is_even_integer()`, `__gamma()`, `__riemann_zeta_m_1_glob()`, and `__sin_pi()`.

Referenced by `__riemann_zeta()`.

9.3.1.240 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_m_1 (_Tp __s)`

Return the Riemann zeta function $\zeta(s) - 1$.

Parameters

<code>__s</code>	The argument $s \neq 1$
------------------	-------------------------

Definition at line 628 of file sf_zeta.tcc.

References `__gnu_cxx::__fp_is_integer()`, `__gamma()`, `__riemann_zeta_m_1_glob()`, `__sin_pi()`, `_S_num_zetam1`, and `_S_zetam1`.

Referenced by `__polylog_exp_neg()`, and `__riemann_zeta()`.

9.3.1.241 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_m_1_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:

$$\zeta(s) = (2\pi)^s \Gamma(1-s) \zeta(1-s) / \pi$$

Definition at line 359 of file sf_zeta.tcc.

Referenced by `__riemann_zeta_glob()`, and `__riemann_zeta_m_1()`.

9.3.1.242 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_product (_Tp __s)`

Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.

$$\zeta(s) = \prod_{i=1}^{\infty} \frac{1}{1 - p_i^{-s}}$$

where p_i are the prime numbers.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } \Re(s) > 1$$

For $\Re(s) < 1$ use the reflection formula:

$$\zeta(s) = (2\pi)^s \Gamma(1-s) \zeta(1-s) / \pi$$

Parameters

<code>__s</code>	The argument
------------------	--------------

Definition at line 460 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta()`.

9.3.1.243 `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:

$$\zeta(s) = (2\pi)^s \Gamma(1-s) \zeta(1-s) / \pi$$

Definition at line 257 of file `sf_zeta.tcc`.

References `__gamma()`, and `__sin_pi()`.

Referenced by `__riemann_zeta()`.

9.3.1.244 `template<typename _Tp> _Tp std::__detail::__rising_factorial (_Tp __a, int __n)`

Return the (upper) Pochhammer function or the rising factorial function. The Pochhammer symbol is defined by

$$a^{\overline{n}} = \Gamma(a + \nu) / \Gamma(\nu) = \prod_{k=0}^{n-1} (a + k), (a)_0 = 1$$

Many notations exist for this function:

$$(a)_{\nu}$$

, (especially in the literature of special functions),

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

Definition at line 3062 of file sf_gamma.tcc.

References `__log_gamma()`, `__log_gamma_sign()`, and `std::__detail::_Factorial_table< _Tp >::__n`.

Referenced by `__log_rising_factorial()`, and `__rising_factorial()`.

9.3.1.245 `template<typename _Tp> _Tp std::__detail::__rising_factorial (_Tp __a, _Tp __nu)`

Return the rising factorial function or the (upper) Pochhammer function. The rising factorial function is defined by

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(\nu)$$

Many notations exist for this function:

$$(a)_{\nu}$$

, (especially in the literature of special functions),

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

Definition at line 3117 of file sf_gamma.tcc.

References `__log_gamma()`, `__log_gamma_sign()`, `std::__detail::_Factorial_table< _Tp >::__n`, and `__rising_factorial()`.

9.3.1.246 `template<typename _Tp> _Tp std::__detail::__sin_pi (_Tp __x)`

Return the reperiodized sine of argument x:

$$\sin_{\pi}(x) = \sin(\pi x)$$

Definition at line 52 of file sf_trig.tcc.

Referenced by `__cos_pi()`, `__cosh_pi()`, `__cyl_bessel_ik()`, `__cyl_bessel_jn()`, `__dirichlet_eta()`, `__gamma_reciprocal()`, `__hankel_debye()`, `__lanczos_log_gamma1p()`, `__log_gamma()`, `__poly_laguerre_large_n()`, `__riemann_zeta()`, `__riemann_zeta_glob()`, `__riemann_zeta_m_1()`, `__riemann_zeta_sum()`, `__sin_pi()`, `__sinc_pi()`, `__sinh_pi()`, and `__spouge_log_gamma1p()`.

9.3.1.247 `template<typename _Tp> std::complex<_Tp> std::__detail::__sin_pi (std::complex<_Tp> __z)`

Return the reperiodized sine of complex argument z :

$$\sin_{\pi}(z) = \sin(\pi z) = \sin_{\pi}(x)\cosh_{\pi}(y) + i\cos_{\pi}(x)\sinh_{\pi}(y)$$

Definition at line 183 of file `sf_trig.tcc`.

References `__cos_pi()`, and `__sin_pi()`.

9.3.1.248 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinc (_Tp __x)`

Return the sinus cardinal function

$$\text{sinc}(x) = \frac{\sin(x)}{x}$$

.

Definition at line 52 of file `sf_cardinal.tcc`.

9.3.1.249 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinc_pi (_Tp __x)`

Return the reperiodized sinus cardinal function

$$\text{sinc}_{\pi}(x) = \frac{\sin(\pi x)}{\pi x}$$

.

Definition at line 72 of file `sf_cardinal.tcc`.

References `__sin_pi()`.

9.3.1.250 `template<typename _Tp> __gnu_cxx::__sincos_t<_Tp> std::__detail::__sincos (_Tp __x) [inline]`

Definition at line 312 of file `sf_trig.tcc`.

Referenced by `__sincos_pi()`.

9.3.1.251 `template<> __gnu_cxx::__sincos_t<float> std::__detail::__sincos (float __x) [inline]`

Definition at line 320 of file `sf_trig.tcc`.

9.3.1.252 `template<> __gnu_cxx::__sincos_t<double> std::__detail::__sincos (double __x) [inline]`

Definition at line 332 of file `sf_trig.tcc`.

9.3.1.253 `template<> __gnu_cxx::__sincos_t<long double> std::__detail::__sincos (long double __x) [inline]`

Definition at line 344 of file sf_trig.tcc.

9.3.1.254 `template<typename _Tp> __gnu_cxx::__sincos_t<_Tp> std::__detail::__sincos_pi (_Tp __x)`

Reperiodized sincos.

Definition at line 356 of file sf_trig.tcc.

References `__gnu_cxx::__sincos_t<_Tp>::__cos_v`, `__gnu_cxx::__sincos_t<_Tp>::__sin_v`, and `__sincos()`.

Referenced by `__polar_pi()`.

9.3.1.255 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__sincosint (_Tp __x)`

This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a `pair`.

The sine integral is defined by:

$$Si(x) = \int_0^x dt \frac{\sin(t)}{t}$$

The cosine integral is defined by:

$$Ci(x) = \gamma_E + \log(x) + \int_0^x dt \frac{\cos(t) - 1}{t}$$

Definition at line 226 of file sf_trigint.tcc.

References `__sincosint_asymp()`, `__sincosint_cont_frac()`, and `__sincosint_series()`.

9.3.1.256 `template<typename _Tp> void std::__detail::__sincosint_asymp (_Tp __t, _Tp &_Si, _Tp &_Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.

The asymptotic series is very good for $x > 50$.

Definition at line 159 of file sf_trigint.tcc.

Referenced by `__sincosint()`.

9.3.1.257 `template<typename _Tp> void std::__detail::__sincosint_cont_frac (_Tp __t, _Tp &_Si, _Tp &_Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.

Definition at line 52 of file sf_trigint.tcc.

Referenced by `__sincosint()`.

9.3.1.258 `template<typename _Tp> void std::__detail::__sincosint_series (_Tp __t, _Tp & _Si, _Tp & _Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.

Definition at line 95 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

9.3.1.259 `template<typename _Tp> _Tp std::__detail::__sinh_pi (_Tp __x)`

Return the reperiodized hyperbolic sine of argument x :

$$\sinh_{\pi}(x) = \sinh(\pi x)$$

Definition at line 83 of file `sf_trig.tcc`.

Referenced by `__sinhc_pi()`.

9.3.1.260 `template<typename _Tp> std::complex<_Tp> std::__detail::__sinh_pi (std::complex<_Tp> __z)`

Return the reperiodized hyperbolic sine of complex argument z :

$$\sinh_{\pi}(z) = \sinh(\pi z) = \sinh_{\pi}(x)\cos_{\pi}(y) + i\cosh_{\pi}(x)\sin_{\pi}(y)$$

Definition at line 205 of file `sf_trig.tcc`.

References `__cos_pi()`, and `__sin_pi()`.

9.3.1.261 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinhc (_Tp __x)`

Return the hyperbolic sinus cardinal function

$$\sinhc(x) = \frac{\sinh(x)}{x}$$

.

Definition at line 97 of file `sf_cardinal.tcc`.

9.3.1.262 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::__detail::__sinhc_pi (_Tp __x)`

Return the reperiodized hyperbolic sinus cardinal function

$$\sinhc_{\pi}(x) = \frac{\sinh(\pi x)}{\pi x}$$

.

Definition at line 115 of file `sf_cardinal.tcc`.

References `__sinh_pi()`.

9.3.1.263 `template<typename _Tp> _Tp std::__detail::__sinhint (const _Tp __x)`

Return the hyperbolic sine integral $Shi(x)$.

The hyperbolic sine integral is given by

$$Shi(x) = (Ei(x) + E_1(x))/2 = (Ei(x) - Ei(-x))/2$$

Parameters

\leftrightarrow _x	The argument of the hyperbolic sine integral function.
-------------------------	--

Returns

The hyperbolic sine integral.

Definition at line 584 of file sf_expint.tcc.

References `__expint_E1()`, and `__expint_Ei()`.

9.3.1.264 `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 754 of file sf_bessel.tcc.

References `__sph_bessel_jn()`.

9.3.1.265 `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 1274 of file sf_hankel.tcc.

References `__sph_hankel()`.

```
9.3.1.266  template<typename _Tp > __gnu_cxx::__sph_mod_bessel_t<unsigned int, _Tp, _Tp>
           std::__detail::__sph_bessel_ik ( unsigned int __n, _Tp __x )
```

Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i'_n(x)$ and $k'_n(x)$ respectively.

Parameters

\leftrightarrow __n	The order of the modified spherical Bessel function.
\leftrightarrow __x	The argument of the modified spherical Bessel function.

Returns

A struct containing the modified spherical Bessel functions of the first and second kinds and their derivatives.

Definition at line 421 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ik()`.

```
9.3.1.267  template<typename _Tp > __gnu_cxx::__sph_bessel_t<unsigned int, _Tp, _Tp> std::__detail::__sph_bessel_jn (
           unsigned int __n, _Tp __x )
```

Compute the spherical Bessel $j_n(x)$ and Neumann $n_n(x)$ functions and their first derivatives $j'_n(x)$ and $n'_n(x)$ respectively.

Parameters

\leftrightarrow __n	The order of the spherical Bessel function.
\leftrightarrow __x	The argument of the spherical Bessel function.

Returns

The output derivative of the spherical Neumann function.

Definition at line 689 of file sf_bessel.tcc.

References `__cyl_bessel_jn()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

9.3.1.268 `template<typename _Tp> __gnu_cxx::__sph_bessel_t<unsigned int, _Tp, std::complex<_Tp>> > std::__detail::__sph_bessel_jn_neg_arg (unsigned int __n, _Tp __x)`

Return the spherical Bessel functions and their derivatives of order ν and argument $x < 0$.

Definition at line 713 of file `sf_bessel.tcc`.



References `__cyl_bessel_jn_neg_arg()`.

Referenced by `__sph_hankel_1()`, and `__sph_hankel_2()`.

9.3.1.269 `template<typename _Tp> __gnu_cxx::__sph_hankel_t<unsigned int, std::complex<_Tp>, std::complex<_Tp>> > std::__detail::__sph_hankel (unsigned int __n, std::complex<_Tp> __z)`

Helper to compute complex spherical Hankel functions and their derivatives.

Parameters

in	 <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.

Returns

A struct containing the spherical Hankel functions of the first and second kinds and their derivatives.

Definition at line 1210 of file `sf_hankel.tcc`.

References `__hankel()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

9.3.1.270 `template<typename _Tp> std::complex<_Tp> std::__detail::__sph_hankel_1 (unsigned int __n, _Tp __x)`

Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.

The spherical Hankel function of the first kind is defined by:

$$h_n^{(1)}(x) = j_n(x) + in_n(x)$$

Parameters

\leftrightarrow _n	The order of the spherical Neumann function.
\leftrightarrow _x	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Definition at line 815 of file sf_bessel.tcc.

References `__sph_bessel_jn()`, and `__sph_bessel_jn_neg_arg()`.

```
9.3.1.271  template<typename _Tp> std::complex<_Tp> std::__detail::__sph_hankel_1 ( unsigned int __n, std::complex<_Tp>
> __z )
```

Return the complex spherical Hankel function of the first kind.

Parameters

in	\leftrightarrow _n	The order for which the spherical Hankel function of the first kind is evaluated.
in	\leftrightarrow _z	The argument at which the spherical Hankel function of the first kind is evaluated.

Returns

The complex spherical Hankel function of the first kind.

Definition at line 1240 of file sf_hankel.tcc.

References `__sph_hankel()`.

```
9.3.1.272  template<typename _Tp> std::complex<_Tp> std::__detail::__sph_hankel_2 ( unsigned int __n, _Tp __x )
```

Return the spherical Hankel function of the second kind $h_n^{(2)}(x)$.

The spherical Hankel function of the second kind is defined by:

$$h_n^{(2)}(x) = j_n(x) - in_n(x)$$

Parameters

\leftrightarrow _n	The non-negative integral order
\leftrightarrow _x	The non-negative real argument

Returns

The output spherical Neumann function.

Definition at line 849 of file sf_bessel.tcc.

References `__sph_bessel_jn()`, and `__sph_bessel_jn_neg_arg()`.

9.3.1.273 `template<typename _Tp> std::complex<_Tp> std::__detail::__sph_hankel_2 (unsigned int __n, std::complex<_Tp> __z)`

Return the complex spherical Hankel function of the second kind.

Parameters

in	\leftrightarrow _n	The order for which the spherical Hankel function of the second kind is evaluated.
in	\leftrightarrow _z	The argument at which the spherical Hankel function of the second kind is evaluated.

Returns

The complex spherical Hankel function of the second kind.

Definition at line 1257 of file sf_hankel.tcc.

References `__sph_hankel()`.

9.3.1.274 `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

<code>__l</code>	The order of the spherical harmonic function. $l \geq 0$.
<code>__m</code>	The order of the spherical harmonic function. $m \leq l$.
<code>__theta</code>	The radian polar angle argument of the spherical harmonic function.
<code>__phi</code>	The radian azimuthal angle argument of the spherical harmonic function.

Definition at line 355 of file `sf_legendre.tcc`.

References `__sph_legendre()`.

9.3.1.275 `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 258 of file `sf_legendre.tcc`.

References `__log_gamma()`, and `__poly_legendre_p()`.

Referenced by `__hydrogen()`, and `__sph_harmonic()`.

9.3.1.276 `template<typename _Tp> _Tp std::__detail::__sph_neumann (unsigned int __n, _Tp __x)`

Return the spherical Neumann function $n_n(x)$ of order n and non-negative real argument x .

The spherical Neumann function is defined by:

$$n_n(x) = \left(\frac{\pi}{2x} \right)^{1/2} N_{n+1/2}(x)$$

Parameters

\leftrightarrow _n	The order of the spherical Neumann function.
\leftrightarrow _x	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Definition at line 787 of file sf_bessel.tcc.

References `__sph_bessel_jn()`.

9.3.1.277 `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 1291 of file sf_hankel.tcc.

References `__sph_hankel()`.

9.3.1.278 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__spouge_binet1p (_Tp __z)`

Return the Binet function $J(1+z)$ by the Spouge method. The Binet function is the log of the scaled Gamma function $\log(\Gamma^*(z))$ defined by

$$J(z) = \log(\Gamma^*(z)) = \log(\Gamma(z)) + z - \left(z - \frac{1}{2}\right) \log(z) - \log(2\pi)$$

or

$$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

where $\Gamma(z)$ is the gamma function.

Parameters

<code>_↔</code>	The argument of the log of the gamma function.
<code>_z</code>	

Returns

The logarithm of the gamma function.

Definition at line 1918 of file `sf_gamma.tcc`.

Referenced by `__spouge_log_gamma1p()`.

9.3.1.279 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__spouge_log_gamma1p (_Tp __z)`

Return the logarithm of the gamma function $\log(\Gamma(1+z))$ by the Spouge algorithm:

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[\sqrt{2\pi} + \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

.

If the argument is real, the log of the absolute value of the Gamma function is returned. The sign to be applied to the exponential of this log Gamma can be recovered with a call to `__log_gamma_sign`.

For complex argument the fully complex log of the gamma function is returned.

See also

Spouge, J. L., Computation of the gamma, digamma, and trigamma functions. *SIAM Journal on Numerical Analysis* 31, 3 (1994), pp. 931-944

Parameters

<code>_↔</code>	The argument of the gamma function.
<code>_z</code>	

Returns

The the gamma function.

Definition at line 1962 of file `sf_gamma.tcc`.

References `__sin_pi()`, and `__spouge_binet1p()`.

Referenced by `__log_gamma()`.

9.3.1.280 `template<typename _Tp> _Tp std::__detail::__stirling_1 (unsigned int __n, unsigned int __m)`

Return the Stirling number of the first kind.

The Stirling numbers of the first kind are the coefficients of the Pochhammer polynomials:

$$(x)_n = \sum_{k=0}^n S_n^{(k)} x^k$$

The recursion is

$$S_{n+1}^{(m)} = S_n^{(m-1)} - n S_n^{(m)} \text{ or}$$

with starting values

$$S_0^{(0 \rightarrow m)} = 1, 0, 0, \dots, 0$$

and

$$S_{0 \rightarrow n}^{(0)} = 1, 0, 0, \dots, 0$$

Todo Find asymptotic solutions for the Stirling numbers of the first kind.

Develop an iterator model for Stirling numbers of the first kind.

Definition at line 300 of file `sf_stirling.tcc`.

9.3.1.281 `template<typename _Tp> _Tp std::__detail::__stirling_1_recur (unsigned int __n, unsigned int __m)`

Return the Stirling number of the first kind by recursion. The recursion is

$$S_{n+1}^{(m)} = S_n^{(m-1)} - n S_n^{(m)} \text{ or}$$

with starting values

$$S_0^{(0 \rightarrow m)} = 1, 0, 0, \dots, 0$$

and

$$S_{0 \rightarrow n}^{(0)} = 1, 0, 0, \dots, 0$$

Definition at line 251 of file `sf_stirling.tcc`.

9.3.1.282 `template<typename _Tp> _Tp std::__detail::__stirling_1_series (unsigned int __n, unsigned int __m)`

Return the Stirling number of the first kind by series expansion. N.B. This seems to be a total disaster.

Definition at line 196 of file `sf_stirling.tcc`.

References `__gnu_cxx::__parity()`.

9.3.1.283 `template<typename _Tp> _Tp std::__detail::__stirling_2 (unsigned int __n, unsigned int __m)`

Return the Stirling number of the second kind from lookup or by series expansion.

The series is:

$$\sigma_n^{(m)} = \sum_{k=0}^m \frac{(-1)^{m-k} k^n}{(m-k)! k!}$$

Todo Find asymptotic solutions for Stirling numbers of the second kind.

Develop an iterator model for Stirling numbers of the second kind.

Definition at line 159 of file sf_stirling.tcc.

9.3.1.284 `template<typename _Tp> _Tp std::__detail::__stirling_2_recur (unsigned int __n, unsigned int __m)`

Return the Stirling number of the second kind by recursion. The recursion is

$$\left\{ \begin{matrix} n \\ m \end{matrix} \right\} = m \left\{ \begin{matrix} n-1 \\ m \end{matrix} \right\} + \left\{ \begin{matrix} n-1 \\ m-1 \end{matrix} \right\}$$

with starting values

$$\left\{ \begin{matrix} 0 \\ 0 \rightarrow m \end{matrix} \right\} = 1, 0, 0, \dots, 0$$

and

$$\left\{ \begin{matrix} 0 \rightarrow n \\ 0 \end{matrix} \right\} = 1, 0, 0, \dots, 0$$

The Stirling number of the second kind is denoted by other symbols in the literature: $\sigma_n^{(m)}$, $S_n^{(m)}$ and others.

Definition at line 122 of file sf_stirling.tcc.

9.3.1.285 `template<typename _Tp> _Tp std::__detail::__stirling_2_series (unsigned int __n, unsigned int __m)`

Return the Stirling number of the second kind from lookup or by series expansion.

The series is:

$$\sigma_n^{(m)} = \left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_{k=0}^m \frac{(-1)^{m-k} k^n}{(m-k)! k!}$$

The Stirling number of the second kind is denoted by other symbols in the literature: $\sigma_n^{(m)}$, $S_n^{(m)}$ and others.

Todo Find a way to predict the maximum Stirling number for a type.

Definition at line 67 of file sf_stirling.tcc.

9.3.1.286 `template<typename _Tp> _Tp std::__detail::__student_t_cdf (_Tp __t, unsigned int __nu)`

Return the Students T probability function.

The students T propability function is related to the incomplete beta function:

$$A(t|\nu) = 1 - I_{\frac{\nu}{\nu+t^2}} \left(\frac{\nu}{2}, \frac{1}{2} \right) A(t|\nu) =$$

Parameters

<u> </u> <i>t</i>	
<u> </u> <i>nu</i>	

Definition at line 444 of file sf_distributions.tcc.

References `__beta_inc()`.

9.3.1.287 `template<typename _Tp> _Tp std::__detail::__student_t_cdfc (_Tp __t, unsigned int __nu)`

Return the complement of the Students T probability function.

The complement of the students T propability function is:

$$A_c(t|\nu) = I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right) = 1 - A(t|\nu)$$

Parameters

<u> </u> <i>t</i>	
<u> </u> <i>nu</i>	

Definition at line 467 of file sf_distributions.tcc.

References `__beta_inc()`.

9.3.1.288 `template<typename _Tp> _Tp std::__detail::__student_t_pdf (_Tp __t, unsigned int __nu)`

Return the Students T probability density.

The students T propability density is:

$$A(t|\nu) = 1 - I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right)A(t|\nu) =$$

Parameters

<u> </u> <i>t</i>	
<u> </u> <i>nu</i>	

Definition at line 419 of file sf_distributions.tcc.

References `__gamma()`.

9.3.1.289 `template<typename _Tp> _Tp std::__detail::__tan_pi (_Tp __x)`

Return the reperiodized tangent of argument x:

$$\tan_{\pi} i(x) = \tan(\pi x)$$

Definition at line 149 of file sf_trig.tcc.

Referenced by `__psi()`, `__tan_pi()`, and `__tanh_pi()`.

9.3.1.290 `template<typename _Tp> std::complex<_Tp> std::__detail::__tan_pi (std::complex<_Tp> __z)`

Return the reperiodized tangent of complex argument z:

$$\tan_{\pi}(z) = \tan(\pi z) = \frac{\tan_{\pi}(x) + i \tanh_{\pi}(y)}{1 - i \tan_{\pi}(x) \tanh_{\pi}(y)}$$

Definition at line 271 of file sf_trig.tcc.

References `__tan_pi()`.

9.3.1.291 `template<typename _Tp> _Tp std::__detail::__tanh_pi (_Tp __x)`

Return the reperiodized hyperbolic tangent of argument x:

$$\tanh_{\pi}(x) = \tanh(\pi x)$$

Definition at line 165 of file sf_trig.tcc.

9.3.1.292 `template<typename _Tp> std::complex<_Tp> std::__detail::__tanh_pi (std::complex<_Tp> __z)`

Return the reperiodized hyperbolic tangent of complex argument z:

$$\tanh_{\pi}(z) = \tanh(\pi z) = \frac{\tanh_{\pi}(x) + i \tan_{\pi}(y)}{1 + i \tanh_{\pi}(x) \tan_{\pi}(y)}$$

Definition at line 294 of file sf_trig.tcc.

References `__tan_pi()`.

9.3.1.293 `template<typename _Tp> _Tp std::__detail::__tgamma (_Tp __a, _Tp __x)`

Return the upper incomplete gamma function. The lower incomplete gamma function is defined by

$$\Gamma(a, x) = \int_x^{\infty} e^{-t} t^{a-1} dt (a > 0)$$

.

Definition at line 2865 of file sf_gamma.tcc.

References `__gnu_cxx::__fp_is_integer()`, `__gamma_cont_frac()`, and `__gamma_series()`.

Referenced by `__gamma_cdf()`.

9.3.1.294 `template<typename _Tp> _Tp std::__detail::__tgamma_lower (_Tp __a, _Tp __x)`

Return the lower incomplete gamma function. The lower incomplete gamma function is defined by

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

.

Definition at line 2830 of file sf_gamma.tcc.

References `__gnu_cxx::__fp_is_integer()`, `__gamma_cont_frac()`, and `__gamma_series()`.

Referenced by `__gamma_cdf()`.

9.3.1.295 `template<typename _Tp> _Tp std::__detail::__theta_1 (_Tp __nu, _Tp __x)`

Return the exponential theta-1 function of period `nu` and argument `x`.

The Neville theta-1 function is defined by

$$\theta_1(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j - 1/2)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 192 of file sf_theta.tcc.

References `__theta_2()`.

Referenced by `__theta_s()`.

9.3.1.296 `template<typename _Tp> _Tp std::__detail::__theta_2 (_Tp __nu, _Tp __x)`

Return the exponential theta-2 function of period `nu` and argument `x`.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 164 of file `sf_theta.tcc`.

References `__theta_2_asymp()`, and `__theta_2_sum()`.

Referenced by `__theta_1()`, and `__theta_c()`.

9.3.1.297 `template<typename _Tp> _Tp std::__detail::__theta_2_asymp (_Tp __nu, _Tp __x)`

Compute and return the θ_2 function by series expansion.

Definition at line 105 of file `sf_theta.tcc`.

Referenced by `__theta_2()`.

9.3.1.298 `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 51 of file `sf_theta.tcc`.

Referenced by `__theta_2()`.

9.3.1.299 `template<typename _Tp> _Tp std::__detail::__theta_3 (_Tp __nu, _Tp __x)`

Return the exponential theta-3 function of period `nu` and argument `x`.

The exponential theta-3 function is defined by

$$\theta_3(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 218 of file sf_theta.tcc.

References `__theta_3_asyp()`, and `__theta_3_sum()`.

Referenced by `__theta_4()`, and `__theta_d()`.

9.3.1.300 `template<typename _Tp> _Tp std::__detail::__theta_3_asyp (_Tp __nu, _Tp __x)`

Compute and return the θ_3 function by asymptotic series expansion.

Definition at line 130 of file sf_theta.tcc.

Referenced by `__theta_3()`.

9.3.1.301 `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 79 of file sf_theta.tcc.

Referenced by `__theta_3()`.

9.3.1.302 `template<typename _Tp> _Tp std::__detail::__theta_4 (_Tp __nu, _Tp __x)`

Return the exponential theta-2 function of period `nu` and argument `x`.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 246 of file sf_theta.tcc.

References `__theta_3()`.

Referenced by `__theta_n()`.

9.3.1.303 `template<typename _Tp> _Tp std::__detail::__theta_c(_Tp __k, _Tp __x)`

Return the Neville θ_c function

$$\theta_c(k, x) = \sqrt{\frac{\pi}{2kK(k)}} \theta_1 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

Definition at line 360 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_2()`.

9.3.1.304 `template<typename _Tp> _Tp std::__detail::__theta_d(_Tp __k, _Tp __x)`

Return the Neville θ_d function

$$\theta_d(k, x) = \sqrt{\frac{\pi}{2K(k)}} \theta_3 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

Definition at line 389 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_3()`.

9.3.1.305 `template<typename _Tp> _Tp std::__detail::__theta_n(_Tp __k, _Tp __x)`

Return the Neville θ_n function

The Neville theta-n function is defined by

$$\theta_n(k, x) = \sqrt{\frac{\pi}{2k'K(k)}} \theta_4 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

Definition at line 420 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_4()`.

9.3.1.306 `template<typename _Tp> _Tp std::__detail::__theta_s(_Tp __k, _Tp __x)`

Return the Neville θ_s function

$$\theta_s(k, x) = \sqrt{\frac{\pi}{2kk'K(k)}} \theta_1 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

Definition at line 330 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_1()`.

9.3.1.307 `template<typename _Tp> _Tp std::__detail::__tricomi_u(_Tp __a, _Tp __c, _Tp __x)`

Return the Tricomi confluent hypergeometric function

$$U(a, c, x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a; c; x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1; 2-c; x)$$

.

Parameters

\leftrightarrow _a	The <i>numerator</i> parameter.
\leftrightarrow _c	The <i>denominator</i> parameter.
\leftrightarrow _x	The argument of the confluent hypergeometric function.

Returns

The Tricomi confluent hypergeometric function.

Definition at line 346 of file sf_hyperg.tcc.

References `__tricomi_u_naive()`.

9.3.1.308 `template<typename _Tp> _Tp std::__detail::__tricomi_u_naive (_Tp __a, _Tp __c, _Tp __x)`

Return the Tricomi confluent hypergeometric function

$$U(a, c, x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a; c; x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1; 2-c; x)$$

.

Parameters

\leftrightarrow _a	The <i>numerator</i> parameter.
\leftrightarrow _c	The <i>denominator</i> parameter.
\leftrightarrow _x	The argument of the confluent hypergeometric function.

Returns

The Tricomi confluent hypergeometric function.

Definition at line 312 of file sf_hyperg.tcc.

References `__conf_hyperg()`, `__gnu_cxx::__fp_is_integer()`, and `__gnu_cxx::tgamma()`.

Referenced by `__tricomi_u()`.

9.3.1.309 `template<typename _Tp> _Tp std::__detail::__weibull_cdf (_Tp __a, _Tp __b, _Tp __x)`

Return the Weibull cumulative probability density function.

The formula for the Weibull cumulative probability density function is

$$F(x|\lambda) = 1 - e^{-(x/b)^a} \text{ for } x \geq 0$$

Definition at line 395 of file `sf_distributions.tcc`.

9.3.1.310 `template<typename _Tp> _Tp std::__detail::__weibull_pdf (_Tp __a, _Tp __b, _Tp __x)`

Return the Weibull probability density function.

The formula for the Weibull probability density function is

$$f(x|a, b) = \frac{a}{b} \left(\frac{x}{b}\right)^{a-1} \exp - \left(\frac{x}{b}\right)^a \text{ for } x \geq 0$$

Definition at line 374 of file `sf_distributions.tcc`.

9.3.1.311 `template<typename _Tp> __gnu_cxx::__promote_fp_t<_Tp> std::__detail::__zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)`

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative integral degree n , signed integral order m , and real radial argument ρ and azimuthal angle ϕ .

The even Zernicke polynomials are defined by:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

and the odd Zernicke polynomials are defined by:

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative degree m and $m \leq n$ and where $R_n^m(\rho)$ is the radial polynomial (

See also

[__poly_radial_jacobi](#)).

Principals of Optics, 7th edition, Max Born and Emil Wolf, Cambridge University Press, 1999, pp 523-525 and 905-910.

Template Parameters

<code>_Tp</code>	The real type of the radial coordinate and azimuthal angle
------------------	--

Parameters

<code>__n</code>	The non-negative integral degree.
<code>__m</code>	The integral azimuthal order
<code>__rho</code>	The radial coordinate
<code>__phi</code>	The azimuthal angle

Definition at line 193 of file sf_jacobi.tcc.

References `__poly_radial_jacobi()`.

9.3.1.312 `template<typename _Tp> _Tp std::__detail::__znorm1 (_Tp __x)`

Definition at line 58 of file sf_owens_t.tcc.

Referenced by `__owens_t()`.

9.3.1.313 `template<typename _Tp> _Tp std::__detail::__znorm2 (_Tp __x)`

Definition at line 47 of file sf_owens_t.tcc.

Referenced by `__owens_t()`.

9.3.2 Variable Documentation

9.3.2.1 `template<typename _Tp> constexpr int std::__detail::__max_FGH = _Airy_series<_Tp>::__N_FGH`

Definition at line 179 of file sf_airy.tcc.

9.3.2.2 `template<> constexpr int std::__detail::__max_FGH< double > = 79`

Definition at line 185 of file sf_airy.tcc.

9.3.2.3 `template<> constexpr int std::__detail::__max_FGH< float > = 15`

Definition at line 182 of file sf_airy.tcc.

9.3.2.4 constexpr size_t std::__detail::_Num_Euler_Maclaurin_zeta = 100

Coefficients for Euler-Maclaurin summation of zeta functions.

$$B_{2j}/(2j)!$$

where B_k are the Bernoulli numbers.

Definition at line 67 of file sf_zeta.tcc.

Referenced by __polylog_exp_neg().

9.3.2.5 constexpr _Factorial_table<long double> std::__detail::_S_double_factorial_table[301]

Definition at line 278 of file sf_gamma.tcc.

Referenced by __double_factorial(), and __log_double_factorial().

9.3.2.6 constexpr long double std::__detail::_S_Euler_Maclaurin_zeta[_Num_Euler_Maclaurin_zeta]

Definition at line 70 of file sf_zeta.tcc.

Referenced by __hurwitz_zeta_euler_maclaurin(), __polylog_exp_neg(), and __riemann_zeta_euler_maclaurin().

9.3.2.7 constexpr _Factorial_table<long double> std::__detail::_S_factorial_table[171]

Definition at line 88 of file sf_gamma.tcc.

Referenced by __factorial(), __gamma(), __gamma_reciprocal(), __log_factorial(), and __log_gamma().

9.3.2.8 constexpr unsigned long long std::__detail::_S_harmonic_denom[_S_num_harmonic_number]

Definition at line 3214 of file sf_gamma.tcc.

Referenced by __harmonic_number().

9.3.2.9 constexpr unsigned long long std::__detail::_S_harmonic_numer[_S_num_harmonic_number]

Definition at line 3181 of file sf_gamma.tcc.

Referenced by __harmonic_number().

9.3.2.10 `constexpr _Factorial_table<long double> std::__detail::_S_neg_double_factorial_table[999]`

Definition at line 599 of file `sf_gamma.tcc`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

9.3.2.11 `template<typename _Tp> constexpr std::size_t std::__detail::_S_num_double_factorials = 0`

Definition at line 263 of file `sf_gamma.tcc`.

9.3.2.12 `template<> constexpr std::size_t std::__detail::_S_num_double_factorials< double > = 301`

Definition at line 268 of file `sf_gamma.tcc`.

9.3.2.13 `template<> constexpr std::size_t std::__detail::_S_num_double_factorials< float > = 57`

Definition at line 266 of file `sf_gamma.tcc`.

9.3.2.14 `template<> constexpr std::size_t std::__detail::_S_num_double_factorials< long double > = 301`

Definition at line 270 of file `sf_gamma.tcc`.

9.3.2.15 `template<typename _Tp> constexpr std::size_t std::__detail::_S_num_factorials = 0`

Definition at line 73 of file `sf_gamma.tcc`.

9.3.2.16 `template<> constexpr std::size_t std::__detail::_S_num_factorials< double > = 171`

Definition at line 78 of file `sf_gamma.tcc`.

9.3.2.17 `template<> constexpr std::size_t std::__detail::_S_num_factorials< float > = 35`

Definition at line 76 of file `sf_gamma.tcc`.

9.3.2.18 `template<> constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171`

Definition at line 80 of file `sf_gamma.tcc`.

9.3.2.19 `constexpr unsigned long long std::__detail::_S_num_harmonic_numer = 29`

Definition at line 3178 of file `sf_gamma.tcc`.

Referenced by `__harmonic_number()`.

9.3.2.20 `template<typename _Tp > constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0`

Definition at line 583 of file `sf_gamma.tcc`.

9.3.2.21 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150`

Definition at line 588 of file `sf_gamma.tcc`.

9.3.2.22 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27`

Definition at line 586 of file `sf_gamma.tcc`.

9.3.2.23 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999`

Definition at line 590 of file `sf_gamma.tcc`.

9.3.2.24 `constexpr size_t std::__detail::_S_num_zetam1 = 121`

Table of $\zeta(n) - 1$ from 0 - 120. MPFR @ 128 bits.

Definition at line 491 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

9.3.2.25 `constexpr long double std::__detail::_S_zetam1[_S_num_zetam1]`

Definition at line 495 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

Chapter 10

Class Documentation

10.1 `__gnu_cxx::__airy_t<_Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of the Airy functions.

Public Attributes

- `_Tp __Ai_deriv`
The derivative of the Airy function Ai.
- `_Tp __Ai_value`
The value of the Airy function Ai.
- `_Tp __Bi_deriv`
The derivative of the Airy function Bi.
- `_Tp __Bi_value`
The value of the Airy function Bi.
- `_Tx __x_arg`
The argument of the Airy functions.

10.1.1 Detailed Description

```
template<typename _Tx, typename _Tp>  
struct __gnu_cxx::__airy_t<_Tx, _Tp>
```

Definition at line 115 of file `specfun_state.h`.

10.1.2 Member Function Documentation

10.1.2.1 `template<typename _Tx, typename _Tp> _Tp __gnu_cxx::__airy_t<_Tx, _Tp>::__Wronskian () const`
`[inline]`

Return the Wronskian of the Airy functions.

Definition at line 133 of file `specfun_state.h`.

10.1.3 Member Data Documentation

10.1.3.1 `template<typename _Tx, typename _Tp> _Tp __gnu_cxx::__airy_t<_Tx, _Tp>::__Ai_deriv`

The derivative of the Airy function Ai.

Definition at line 124 of file `specfun_state.h`.

10.1.3.2 `template<typename _Tx, typename _Tp> _Tp __gnu_cxx::__airy_t<_Tx, _Tp>::__Ai_value`

The value of the Airy function Ai.

Definition at line 121 of file `specfun_state.h`.

10.1.3.3 `template<typename _Tx, typename _Tp> _Tp __gnu_cxx::__airy_t<_Tx, _Tp>::__Bi_deriv`

The derivative of the Airy function Bi.

Definition at line 130 of file `specfun_state.h`.

10.1.3.4 `template<typename _Tx, typename _Tp> _Tp __gnu_cxx::__airy_t<_Tx, _Tp>::__Bi_value`

The value of the Airy function Bi.

Definition at line 127 of file `specfun_state.h`.

10.1.3.5 `template<typename _Tx, typename _Tp> _Tx __gnu_cxx::__airy_t<_Tx, _Tp>::__x_arg`

The argument of the Airy functions.

Definition at line 118 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.2 `__gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of the cylindrical Bessel functions.

Public Attributes

- `_Tp __J_deriv`
The derivative of the Bessel function of the first kind.
- `_Tp __J_value`
The value of the Bessel function of the first kind.
- `_Tp __N_deriv`
The derivative of the Bessel function of the second kind.
- `_Tp __N_value`
The value of the Bessel function of the second kind.
- `_Tnu __nu_arg`
The real order of the cylindrical Bessel functions.
- `_Tx __x_arg`
The argument of the cylindrical Bessel functions.

10.2.1 Detailed Description

```
template<typename _Tnu, typename _Tx, typename _Tp>
struct __gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp>
```

This struct captures the state of the cylindrical Bessel functions at a given order and argument.

Definition at line 168 of file `specfun_state.h`.

10.2.2 Member Function Documentation

10.2.2.1 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp>::__Wronskian () const` `[inline]`

Return the Wronskian of the cylindrical Bessel functions.

Definition at line 189 of file `specfun_state.h`.

10.2.3 Member Data Documentation

10.2.3.1 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp>::__J_deriv`

The derivative of the Bessel function of the first kind.

Definition at line 180 of file `specfun_state.h`.

10.2.3.2 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp>::__J_value`

The value of the Bessel function of the first kind.

Definition at line 177 of file `specfun_state.h`.

10.2.3.3 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp>::__N_deriv`

The derivative of the Bessel function of the second kind.

Definition at line 186 of file `specfun_state.h`.

10.2.3.4 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp>::__N_value`

The value of the Bessel function of the second kind.

Definition at line 183 of file `specfun_state.h`.

10.2.3.5 `template<typename _Tnu, typename _Tx, typename _Tp> _Tnu __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp>::__nu_arg`

The real order of the cylindrical Bessel functions.

Definition at line 171 of file `specfun_state.h`.

10.2.3.6 `template<typename _Tnu, typename _Tx, typename _Tp> _Tx __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp>::__x_arg`

The argument of the cylindrical Bessel functions.

Definition at line 174 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.3 `__gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of the cylindrical Hankel functions.

Public Attributes

- `_Tp __H1_deriv`
The derivative of the cylindrical Hankel function of the first kind.
- `_Tp __H1_value`
The value of the cylindrical Hankel function of the first kind.
- `_Tp __H2_deriv`
The derivative of the cylindrical Hankel function of the second kind.
- `_Tp __H2_value`
The value of the cylindrical Hankel function of the second kind.
- `_Tnu __nu_arg`
The real order of the cylindrical Hankel functions.
- `_Tx __x_arg`
The argument of the modified Hankel functions.

10.3.1 Detailed Description

```
template<typename _Tnu, typename _Tx, typename _Tp>
struct __gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp>
```

`_Tp` pretty much has to be complex.

Definition at line 231 of file `specfun_state.h`.

10.3.2 Member Function Documentation

10.3.2.1 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp>::__Wronskian () const` `[inline]`

Return the Wronskian of the cylindrical Hankel functions.

Definition at line 252 of file `specfun_state.h`.

10.3.3 Member Data Documentation

10.3.3.1 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp>::__H1_deriv`

The derivative of the cylindrical Hankel function of the first kind.

Definition at line 243 of file `specfun_state.h`.

10.3.3.2 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp>::__H1_value`

The value of the cylindrical Hankel function of the first kind.

Definition at line 240 of file `specfun_state.h`.

10.3.3.3 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp>::__H2_deriv`

The derivative of the cylindrical Hankel function of the second kind.

Definition at line 249 of file `specfun_state.h`.

10.3.3.4 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp>::__H2_value`

The value of the cylindrical Hankel function of the second kind.

Definition at line 246 of file `specfun_state.h`.

10.3.3.5 `template<typename _Tnu, typename _Tx, typename _Tp> _Tnu __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp>::__nu_arg`

The real order of the cylindrical Hankel functions.

Definition at line 234 of file `specfun_state.h`.

10.3.3.6 `template<typename _Tnu, typename _Tx, typename _Tp> _Tx __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp>::__x_arg`

The argument of the modified Hankel functions.

Definition at line 237 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.4 `__gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of the modified cylindrical Bessel functions.

Public Attributes

- `_Tp __I_deriv`
The derivative of the modified cylindrical Bessel function of the first kind.
- `_Tp __I_value`
The value of the modified cylindrical Bessel function of the first kind.
- `_Tp __K_deriv`
The derivative of the modified cylindrical Bessel function of the second kind.
- `_Tp __K_value`
The value of the modified cylindrical Bessel function of the second kind.
- `_Tnu __nu_arg`
The real order of the modified cylindrical Bessel functions.
- `_Tx __x_arg`
The argument of the modified cylindrical Bessel functions.

10.4.1 Detailed Description

```
template<typename _Tnu, typename _Tx, typename _Tp>
struct __gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp>
```

This struct captures the state of the modified cylindrical Bessel functions at a given order and argument.

Definition at line 198 of file `specfun_state.h`.

10.4.2 Member Function Documentation

10.4.2.1 `template<typename _Tnu, typename _Tx, typename _Tp> _Tp __gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp>::__Wronskian () const` `[inline]`

Return the Wronskian of the modified cylindrical Bessel functions.

Definition at line 223 of file `specfun_state.h`.

10.4.3 Member Data Documentation

10.4.3.1 `template<typename _Tnu, typename _Tx, typename _Tp > _Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__I_deriv`

The derivative of the modified cylindrical Bessel function of the first kind.

Definition at line 212 of file `specfun_state.h`.

10.4.3.2 `template<typename _Tnu, typename _Tx, typename _Tp > _Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__I_value`

The value of the modified cylindrical Bessel function of the first kind.

Definition at line 208 of file `specfun_state.h`.

10.4.3.3 `template<typename _Tnu, typename _Tx, typename _Tp > _Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__K_deriv`

The derivative of the modified cylindrical Bessel function of the second kind.

Definition at line 220 of file `specfun_state.h`.

10.4.3.4 `template<typename _Tnu, typename _Tx, typename _Tp > _Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__K_value`

The value of the modified cylindrical Bessel function of the second kind.

Definition at line 216 of file `specfun_state.h`.

10.4.3.5 `template<typename _Tnu, typename _Tx, typename _Tp > _Tnu __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__nu_arg`

The real order of the modified cylindrical Bessel functions.

Definition at line 201 of file `specfun_state.h`.

10.4.3.6 `template<typename _Tnu, typename _Tx, typename _Tp > _Tx __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__x_arg`

The argument of the modified cylindrical Bessel functions.

Definition at line 204 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.5 `__gnu_cxx::__fock_airy_t<_Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of the Fock-type Airy functions.

Public Attributes

- `_Tp __w1_deriv`
The derivative of the Fock-type Airy function w1.
- `_Tp __w1_value`
The value of the Fock-type Airy function w1.
- `_Tp __w2_deriv`
The derivative of the Fock-type Airy function w2.
- `_Tp __w2_value`
The value of the Fock-type Airy function w2.
- `_Tx __x_arg`
The argument of the Fock-type Airy functions.

10.5.1 Detailed Description

```
template<typename _Tx, typename _Tp>
struct __gnu_cxx::__fock_airy_t<_Tx, _Tp>
```

`_Tp` pretty much has to be complex.

Definition at line 141 of file `specfun_state.h`.

10.5.2 Member Function Documentation

10.5.2.1 `template<typename _Tx, typename _Tp> _Tp __gnu_cxx::__fock_airy_t<_Tx, _Tp>::__Wronskian () const`
`[inline]`

Return the Wronskian of the Fock-type Airy functions.

Definition at line 159 of file `specfun_state.h`.

10.5.3 Member Data Documentation

10.5.3.1 `template<typename _Tx, typename _Tp> _Tp __gnu_cxx::__fock_airy_t<_Tx, _Tp>::__w1_deriv`

The derivative of the Fock-type Airy function w1.

Definition at line 150 of file `specfun_state.h`.

10.5.3.2 `template<typename _Tx, typename _Tp> _Tp __gnu_cxx::__fock_airy_t<_Tx, _Tp>::__w1_value`

The value of the Fock-type Airy function w1.

Definition at line 147 of file `specfun_state.h`.

10.5.3.3 `template<typename _Tx, typename _Tp> _Tp __gnu_cxx::__fock_airy_t<_Tx, _Tp>::__w2_deriv`

The derivative of the Fock-type Airy function w2.

Definition at line 156 of file `specfun_state.h`.

10.5.3.4 `template<typename _Tx, typename _Tp> _Tp __gnu_cxx::__fock_airy_t<_Tx, _Tp>::__w2_value`

The value of the Fock-type Airy function w2.

Definition at line 153 of file `specfun_state.h`.

10.5.3.5 `template<typename _Tx, typename _Tp> _Tx __gnu_cxx::__fock_airy_t<_Tx, _Tp>::__x_arg`

The argument of the Fock-type Airy functions.

Definition at line 144 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.6 `__gnu_cxx::__fp_is_integer_t` Struct Reference

```
#include <math_util.h>
```

Public Member Functions

- `operator bool` () const
- `int operator()` () const

Public Attributes

- `bool __is_integral`
- `int __value`

10.6.1 Detailed Description

A struct returned by floating point integer queries.

Definition at line 123 of file `math_util.h`.

10.6.2 Member Function Documentation

10.6.2.1 `__gnu_cxx::__fp_is_integer_t::operator bool` () const `[inline]`

Definition at line 132 of file `math_util.h`.

References `__is_integral`.

10.6.2.2 `int __gnu_cxx::__fp_is_integer_t::operator()` () const `[inline]`

Definition at line 137 of file `math_util.h`.

References `__value`.

10.6.3 Member Data Documentation

10.6.3.1 `bool __gnu_cxx::__fp_is_integer_t::__is_integral`

Definition at line 126 of file `math_util.h`.

Referenced by `operator bool()`.

10.6.3.2 `int __gnu_cxx::__fp_is_integer_t::__value`

Definition at line 129 of file `math_util.h`.

Referenced by `operator()()`.

The documentation for this struct was generated from the following file:

- [ext/math_util.h](#)

10.7 `__gnu_cxx::__gamma_inc_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Attributes

- `_Tp __lgamma_value`
The value of the log of the incomplete gamma function.
- `_Tp __tgamma_value`
The value of the total gamma function.

10.7.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__gamma_inc_t<_Tp>
```

The sign of the exponentiated `log(gamma)` is applied to the `tgamma` value.

Definition at line 369 of file `specfun_state.h`.

10.7.2 Member Data Documentation

10.7.2.1 `template<typename _Tp> _Tp __gnu_cxx::__gamma_inc_t<_Tp>::__lgamma_value`

The value of the log of the incomplete gamma function.

Definition at line 374 of file `specfun_state.h`.

10.7.2.2 `template<typename _Tp> _Tp __gnu_cxx::__gamma_inc_t<_Tp>::__tgamma_value`

The value of the total gamma function.

Definition at line 372 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.8 `__gnu_cxx::__gamma_temme_t<_Tp>` Struct Template Reference

A structure for the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ 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.

```
#include <specfun_state.h>
```

Public Attributes

- `_Tp __gamma_1_value`
The output function $\Gamma_1(\mu)$.
- `_Tp __gamma_2_value`
The output function $\Gamma_2(\mu)$.
- `_Tp __gamma_minus_value`
The output function $1/\Gamma(1-\mu)$.
- `_Tp __gamma_plus_value`
The output function $1/\Gamma(1+\mu)$.
- `_Tp __mu_arg`
The input parameter of the gamma functions.

10.8.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__gamma_temme_t<_Tp>
```

A structure for the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ 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 high for $|\mu| < 0$.

Definition at line 397 of file `specfun_state.h`.

10.8.2 Member Data Documentation

10.8.2.1 `template<typename _Tp> _Tp __gnu_cxx::__gamma_temme_t<_Tp>::__gamma_1_value`

The output function $\Gamma_1(\mu)$.

Definition at line 409 of file `specfun_state.h`.

10.8.2.2 `template<typename _Tp> _Tp __gnu_cxx::__gamma_temme_t<_Tp>::__gamma_2_value`

The output function $\Gamma_2(\mu)$.

Definition at line 412 of file `specfun_state.h`.

10.8.2.3 `template<typename _Tp> _Tp __gnu_cxx::__gamma_temme_t<_Tp>::__gamma_minus_value`

The output function $1/\Gamma(1 - \mu)$.

Definition at line 406 of file `specfun_state.h`.

10.8.2.4 `template<typename _Tp> _Tp __gnu_cxx::__gamma_temme_t<_Tp>::__gamma_plus_value`

The output function $1/\Gamma(1 + \mu)$.

Definition at line 403 of file `specfun_state.h`.

10.8.2.5 `template<typename _Tp> _Tp __gnu_cxx::__gamma_temme_t<_Tp>::__mu_arg`

The input parameter of the gamma functions.

Definition at line 400 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.9 `__gnu_cxx::__jacobi_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __am()` const
- `_Tp __cd()` const
- `_Tp __cs()` const
- `_Tp __dc()` const
- `_Tp __ds()` const
- `_Tp __nc()` const
- `_Tp __nd()` const
- `_Tp __ns()` const
- `_Tp __sc()` const
- `_Tp __sd()` const

Public Attributes

- `_Tp __cn_value`
Jacobi cosine amplitude value.
- `_Tp __dn_value`
Jacobi delta amplitude value.
- `_Tp __sn_value`
Jacobi sine amplitude value.

10.9.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__jacobi_t<_Tp>
```

Definition at line 72 of file `specfun_state.h`.

10.9.2 Member Function Documentation

10.9.2.1 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__am() const` `[inline]`

Definition at line 83 of file `specfun_state.h`.

10.9.2.2 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__cd() const` `[inline]`

Definition at line 101 of file `specfun_state.h`.

10.9.2.3 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__cs() const` `[inline]`

Definition at line 104 of file `specfun_state.h`.

10.9.2.4 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__dc() const` `[inline]`

Definition at line 110 of file `specfun_state.h`.

10.9.2.5 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__ds() const` `[inline]`

Definition at line 107 of file `specfun_state.h`.

10.9.2.6 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__nc() const` `[inline]`

Definition at line 89 of file `specfun_state.h`.

10.9.2.7 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__nd() const` `[inline]`

Definition at line 92 of file `specfun_state.h`.

10.9.2.8 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__ns() const` `[inline]`

Definition at line 86 of file `specfun_state.h`.

10.9.2.9 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__sc() const` `[inline]`

Definition at line 95 of file `specfun_state.h`.

10.9.2.10 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__sd() const` `[inline]`

Definition at line 98 of file `specfun_state.h`.

10.9.3 Member Data Documentation

10.9.3.1 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__cn_value`

Jacobi cosine amplitude value.

Definition at line 78 of file `specfun_state.h`.

10.9.3.2 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__dn_value`

Jacobi delta amplitude value.

Definition at line 81 of file `specfun_state.h`.

10.9.3.3 `template<typename _Tp> _Tp __gnu_cxx::__jacobi_t<_Tp>::__sn_value`

Jacobi sine amplitude value.

Definition at line 75 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.10 __gnu_cxx::__lgamma_t<_Tp> Struct Template Reference

```
#include <specfun_state.h>
```

Public Attributes

- `int __lgamma_sign`
The sign of the exponent of the log gamma value.
- `_Tp __lgamma_value`
The value log gamma function.

10.10.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__lgamma_t<_Tp>
```

The log of the absolute value of the gamma function The sign of the exponentiated log(gamma) is stored in sign.

Definition at line 356 of file `specfun_state.h`.

10.10.2 Member Data Documentation

10.10.2.1 `template<typename _Tp> int __gnu_cxx::__lgamma_t<_Tp>::__lgamma_sign`

The sign of the exponent of the log gamma value.

Definition at line 362 of file `specfun_state.h`.

10.10.2.2 `template<typename _Tp> _Tp __gnu_cxx::__lgamma_t<_Tp>::__lgamma_value`

The value log gamma function.

Definition at line 359 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.11 `__gnu_cxx::__pqgamma_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Attributes

- [_Tp __pgamma_value](#)
- [_Tp __qgamma_value](#)

10.11.1 Detailed Description

```
template<typename _Tp>  
struct __gnu_cxx::__pqgamma_t<_Tp>
```

Definition at line 342 of file `specfun_state.h`.

10.11.2 Member Data Documentation

10.11.2.1 `template<typename _Tp> _Tp __gnu_cxx::__pqgamma_t<_Tp>::__pgamma_value`

Definition at line 345 of file `specfun_state.h`.

10.11.2.2 `template<typename _Tp> _Tp __gnu_cxx::__pqgamma_t<_Tp>::__qgamma_value`

Definition at line 348 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.12 `__gnu_cxx::__quadrature_point_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `__quadrature_point_t()`=default
- `__quadrature_point_t`(`_Tp __z`, `_Tp __w`)

Public Attributes

- `_Tp __weight`
- `_Tp __zero`

10.12.1 Detailed Description

```
template<typename _Tp>  
struct __gnu_cxx::__quadrature_point_t<_Tp>
```

A struct to store a cosine and a sine value. A return for sincos-type functions.

Definition at line 46 of file `specfun_state.h`.

10.12.2 Constructor & Destructor Documentation

10.12.2.1 `template<typename _Tp> __gnu_cxx::__quadrature_point_t<_Tp>::__quadrature_point_t()`
[default]

10.12.2.2 `template<typename _Tp> __gnu_cxx::__quadrature_point_t<_Tp>::__quadrature_point_t(_Tp __z, _Tp __w)` [inline]

Definition at line 53 of file `specfun_state.h`.

10.12.3 Member Data Documentation

10.12.3.1 `template<typename _Tp> _Tp __gnu_cxx::__quadrature_point_t<_Tp>::__weight`

Definition at line 49 of file `specfun_state.h`.

10.12.3.2 `template<typename _Tp> _Tp __gnu_cxx::__quadrature_point_t<_Tp>::__zero`

Definition at line 48 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.13 `__gnu_cxx::__sincos_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Attributes

- `_Tp __cos_v`
- `_Tp __sin_v`

10.13.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__sincos_t<_Tp>
```

A struct to store a cosine and a sine value. A return for sincos-type functions.

Definition at line 64 of file `specfun_state.h`.

10.13.2 Member Data Documentation

10.13.2.1 `template<typename _Tp> _Tp __gnu_cxx::__sincos_t<_Tp>::__cos_v`

Definition at line 67 of file `specfun_state.h`.

Referenced by `std::__detail::__polar_pi()`, and `std::__detail::__sincos_pi()`.

10.13.2.2 `template<typename _Tp> _Tp __gnu_cxx::__sincos_t<_Tp>::__sin_v`

Definition at line 66 of file `specfun_state.h`.

Referenced by `std::__detail::__polar_pi()`, and `std::__detail::__sincos_pi()`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.14 __gnu_cxx::__sph_bessel_t<_Tn, _Tx, _Tp> Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- [_Tp __Wronskian\(\)](#) const
Return the Wronskian of the spherical Bessel functions.

Public Attributes

- [_Tp __j_deriv](#)
The derivative of the spherical Bessel function of the first kind.
- [_Tp __j_value](#)
The value of the spherical Bessel function of the first kind.
- [_Tn __n_arg](#)
The integral order of the spherical Bessel functions.
- [_Tp __n_deriv](#)
The derivative of the spherical Bessel function of the second kind.
- [_Tp __n_value](#)
The value of the spherical Bessel function of the second kind.
- [_Tx __x_arg](#)
The argument of the spherical Bessel functions.

10.14.1 Detailed Description

```
template<typename _Tn, typename _Tx, typename _Tp>
struct __gnu_cxx::__sph_bessel_t<_Tn, _Tx, _Tp>
```

Definition at line 257 of file specfun_state.h.

10.14.2 Member Function Documentation

10.14.2.1 `template<typename _Tn, typename _Tx, typename _Tp> _Tp __gnu_cxx::__sph_bessel_t<_Tn, _Tx, _Tp>::__Wronskian() const` `[inline]`

Return the Wronskian of the spherical Bessel functions.

Definition at line 278 of file specfun_state.h.

10.14.3 Member Data Documentation

10.14.3.1 `template<typename _Tn, typename _Tx, typename _Tp> _Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp>::__j_deriv`

The derivative of the spherical Bessel function of the first kind.

Definition at line 269 of file `specfun_state.h`.

10.14.3.2 `template<typename _Tn, typename _Tx, typename _Tp> _Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp>::__j_value`

The value of the spherical Bessel function of the first kind.

Definition at line 266 of file `specfun_state.h`.

10.14.3.3 `template<typename _Tn, typename _Tx, typename _Tp> _Tn __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp>::__n_arg`

The integral order of the spherical Bessel functions.

Definition at line 260 of file `specfun_state.h`.

10.14.3.4 `template<typename _Tn, typename _Tx, typename _Tp> _Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp>::__n_deriv`

The derivative of the spherical Bessel function of the second kind.

Definition at line 275 of file `specfun_state.h`.

10.14.3.5 `template<typename _Tn, typename _Tx, typename _Tp> _Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp>::__n_value`

The value of the spherical Bessel function of the second kind.

Definition at line 272 of file `specfun_state.h`.

10.14.3.6 `template<typename _Tn, typename _Tx, typename _Tp> _Tx __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp>::__x_arg`

The argument of the spherical Bessel functions.

Definition at line 263 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.15 `__gnu_cxx::__sph_hankel_t<_Tn, _Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of the cylindrical Hankel functions.

Public Attributes

- `_Tp __h1_deriv`
The derivative of the spherical Hankel function of the first kind.
- `_Tp __h1_value`
The value of the spherical Hankel function of the first kind.
- `_Tp __h2_deriv`
The derivative of the spherical Hankel function of the second kind.
- `_Tp __h2_value`
The value of the spherical Hankel function of the second kind.
- `_Tn __n_arg`
The integral order of the spherical Hankel functions.
- `_Tx __x_arg`
The argument of the spherical Hankel functions.

10.15.1 Detailed Description

```
template<typename _Tn, typename _Tx, typename _Tp>
struct __gnu_cxx::__sph_hankel_t<_Tn, _Tx, _Tp>
```

`_Tp` pretty much has to be complex.

Definition at line 316 of file `specfun_state.h`.

10.15.2 Member Function Documentation

10.15.2.1 `template<typename _Tn, typename _Tx, typename _Tp> _Tp __gnu_cxx::__sph_hankel_t<_Tn, _Tx, _Tp>::__Wronskian () const` `[inline]`

Return the Wronskian of the cylindrical Hankel functions.

Definition at line 337 of file `specfun_state.h`.

10.15.3 Member Data Documentation

10.15.3.1 `template<typename _Tn , typename _Tx , typename _Tp > _Tp __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__h1_deriv`

The derivative of the spherical Hankel function of the first kind.

Definition at line 328 of file `specfun_state.h`.

10.15.3.2 `template<typename _Tn , typename _Tx , typename _Tp > _Tp __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__h1_value`

The value of the spherical Hankel function of the first kind.

Definition at line 325 of file `specfun_state.h`.

10.15.3.3 `template<typename _Tn , typename _Tx , typename _Tp > _Tp __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__h2_deriv`

The derivative of the spherical Hankel function of the second kind.

Definition at line 334 of file `specfun_state.h`.

10.15.3.4 `template<typename _Tn , typename _Tx , typename _Tp > _Tp __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__h2_value`

The value of the spherical Hankel function of the second kind.

Definition at line 331 of file `specfun_state.h`.

10.15.3.5 `template<typename _Tn , typename _Tx , typename _Tp > _Tn __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__n_arg`

The integral order of the spherical Hankel functions.

Definition at line 319 of file `specfun_state.h`.

10.15.3.6 `template<typename _Tn , typename _Tx , typename _Tp > _Tx __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__x_arg`

The argument of the spherical Hankel functions.

Definition at line 322 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.16 `__gnu_cxx::__sph_mod_bessel_t<_Tn, _Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of the modified cylindrical Bessel functions.

Public Attributes

- `_Tp __i_deriv`
The derivative of the modified spherical Bessel function of the first kind.
- `_Tp __i_value`
The value of the modified spherical Bessel function of the first kind.
- `_Tp __k_deriv`
The derivative of the modified spherical Bessel function of the second kind.
- `_Tp __k_value`
The value of the modified spherical Bessel function of the second kind.
- `_Tx __x_arg`
The argument of the modified spherical Bessel functions.
- `_Tn n_arg`
The integral order of the modified spherical Bessel functions.

10.16.1 Detailed Description

```
template<typename _Tn, typename _Tx, typename _Tp>
struct __gnu_cxx::__sph_mod_bessel_t<_Tn, _Tx, _Tp>
```

Definition at line 283 of file `specfun_state.h`.

10.16.2 Member Function Documentation

10.16.2.1 `template<typename _Tn, typename _Tx, typename _Tp> _Tp __gnu_cxx::__sph_mod_bessel_t<_Tn, _Tx, _Tp>::__Wronskian () const` `[inline]`

Return the Wronskian of the modified cylindrical Bessel functions.

Definition at line 308 of file `specfun_state.h`.

10.16.3 Member Data Documentation

10.16.3.1 `template<typename _Tn, typename _Tx, typename _Tp > _Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__i_deriv`

The derivative of the modified spherical Bessel function of the first kind.

Definition at line 297 of file `specfun_state.h`.

10.16.3.2 `template<typename _Tn, typename _Tx, typename _Tp > _Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__i_value`

The value of the modified spherical Bessel function of the first kind.

Definition at line 293 of file `specfun_state.h`.

10.16.3.3 `template<typename _Tn, typename _Tx, typename _Tp > _Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__k_deriv`

The derivative of the modified spherical Bessel function of the second kind.

Definition at line 305 of file `specfun_state.h`.

10.16.3.4 `template<typename _Tn, typename _Tx, typename _Tp > _Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__k_value`

The value of the modified spherical Bessel function of the second kind.

Definition at line 301 of file `specfun_state.h`.

10.16.3.5 `template<typename _Tn, typename _Tx, typename _Tp > _Tx __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__x_arg`

The argument of the modified spherical Bessel functions.

Definition at line 286 of file `specfun_state.h`.

10.16.3.6 `template<typename _Tn, typename _Tx, typename _Tp > _Tn __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__n_arg`

The integral order of the modified spherical Bessel functions.

Definition at line 289 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [bits/specfun_state.h](#)

10.17 std::__detail::__gamma_lanczos_data< _Tp > Struct Template Reference

10.17.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::__gamma_lanczos_data< _Tp >
```

A struct for Lanczos algorithm Chebyshev arrays of coefficients.

Definition at line 1995 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- [bits/sf_gamma.tcc](#)

10.18 std::__detail::__gamma_lanczos_data< double > Struct Template Reference

Static Public Attributes

- static constexpr std::array< double, 10 > [_S_cheby](#)
- static constexpr double [_S_g](#) = 9.5

10.18.1 Detailed Description

```
template<>
struct std::__detail::__gamma_lanczos_data< double >
```

Definition at line 2017 of file sf_gamma.tcc.

10.18.2 Member Data Documentation

10.18.2.1 constexpr std::array<double, 10> std::__detail::__gamma_lanczos_data< double >::_S_cheby [static]

Initial value:

```
{
    5.557569219204146e+03,
    -4.248114953727554e+03,
    1.881719608233706e+03,
    -4.705537221412237e+02,
    6.325224688788239e+01,
    -4.206901076213398e+00,
    1.202512485324405e-01,
    -1.141081476816908e-03,
    2.055079676210880e-06,
    1.280568540096283e-09,
}
```

Definition at line 2022 of file sf_gamma.tcc.

10.18.2.2 constexpr double std::__detail::__gamma_lanczos_data< double >::_S_g = 9.5 [static]

Definition at line 2019 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.19 std::__detail::__gamma_lanczos_data< float > Struct Template Reference

Static Public Attributes

- static constexpr std::array< float, 7 > _S_cheby
- static constexpr float _S_g = 6.5F

10.19.1 Detailed Description

```
template<>
struct std::__detail::__gamma_lanczos_data< float >
```

Definition at line 2000 of file sf_gamma.tcc.

10.19.2 Member Data Documentation

10.19.2.1 constexpr std::array<float, 7> std::__detail::__gamma_lanczos_data< float >::_S_cheby [static]

Initial value:

```
{
    3.307139e+02F,
    -2.255998e+02F,
    6.989520e+01F,
    -9.058929e+00F,
    4.110107e-01F,
    -4.150391e-03F,
    -3.417969e-03F,
}
```

Definition at line 2005 of file sf_gamma.tcc.

10.19.2.2 constexpr float std::__detail::__gamma_lanczos_data< float >::_S_g = 6.5F [static]

Definition at line 2002 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.20 std::__detail::__gamma_lanczos_data< long double > Struct Template Reference

Static Public Attributes

- static constexpr std::array< long double, 11 > [_S_cheby](#)
- static constexpr long double [_S_g](#) = 10.5L

10.20.1 Detailed Description

```
template<>
struct std::__detail::__gamma_lanczos_data< long double >
```

Definition at line 2037 of file sf_gamma.tcc.

10.20.2 Member Data Documentation

10.20.2.1 constexpr std::array<long double, 11> std::__detail::__gamma_lanczos_data< long double >::__S_cheby
[static]

Initial value:

```
{
    1.440399692024250728e+04L,
    -1.128006201837065341e+04L,
    5.384108670160999829e+03L,
    -1.536234184127325861e+03L,
    2.528551924697309561e+02L,
    -2.265389090278717887e+01L,
    1.006663776178612579e+00L,
    -1.900805731354182626e-02L,
    1.150508317664389324e-04L,
    -1.208915136885480024e-07L,
    -1.518856151960790157e-10L,
}
```

Definition at line 2042 of file sf_gamma.tcc.

10.20.2.2 constexpr long double std::__detail::__gamma_lanczos_data< long double >::__S_g = 10.5L [static]

Definition at line 2039 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/[sf_gamma.tcc](#)

10.21 `std::__detail::__gamma_spouge_data< _Tp >` Struct Template Reference

10.21.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::__gamma_spouge_data< _Tp >
```

A struct for Spouge algorithm Chebyshev arrays of coefficients.

Definition at line 1769 of file `sf_gamma.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf_gamma.tcc](#)

10.22 `std::__detail::__gamma_spouge_data< double >` Struct Template Reference

Static Public Attributes

- static constexpr `std::array< double, 18 >` `_S_cheby`

10.22.1 Detailed Description

```
template<>
struct std::__detail::__gamma_spouge_data< double >
```

Definition at line 1790 of file `sf_gamma.tcc`.

10.22.2 Member Data Documentation

10.22.2.1 `constexpr std::array<double, 18> std::__detail::__gamma_spouge_data< double >::_S_cheby` `[static]`

Initial value:

```
{
    2.785716565770350e+08,
    -1.693088166941517e+09,
    4.549688586500031e+09,
    -7.121728036151557e+09,
    7.202572947273274e+09,
    -4.935548868770376e+09,
    2.338187776097503e+09,
    -7.678102458920741e+08,
    1.727524819329867e+08,
    -2.595321377008346e+07,
    2.494811203993971e+06,
    -1.437252641338402e+05,
    4.490767356961276e+03,
    -6.505596924745029e+01,
    3.362323142416327e-01,
    -3.817361443986454e-04,
    3.273137866873352e-08,
    -7.642333165976788e-15,
}
```

Definition at line 1794 of file `sf_gamma.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf_gamma.tcc](#)

10.23 std::__detail::__gamma_spouge_data< float > Struct Template Reference

Static Public Attributes

- static constexpr std::array< float, 7 > [_S_cheby](#)

10.23.1 Detailed Description

```
template<>
struct std::__detail::__gamma_spouge_data< float >
```

Definition at line 1774 of file sf_gamma.tcc.

10.23.2 Member Data Documentation

10.23.2.1 constexpr std::array<float, 7> std::__detail::__gamma_spouge_data< float >::_S_cheby [static]

Initial value:

```
{
    2.901419e+03F,
    -5.929168e+03F,
    4.148274e+03F,
    -1.164761e+03F,
    1.174135e+02F,
    -2.786588e+00F,
    3.775392e-03F,
}
```

Definition at line 1778 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- bits/[sf_gamma.tcc](#)

10.24 std::__detail::__gamma_spouge_data< long double > Struct Template Reference

Static Public Attributes

- static constexpr std::array< long double, 22 > [_S_cheby](#)

10.24.1 Detailed Description

```
template<>
struct std::__detail::__gamma_spouge_data< long double >
```

Definition at line 1817 of file sf_gamma.tcc.

10.24.2 Member Data Documentation

10.24.2.1 `constexpr std::array<long double, 22> std::__detail::__gamma_spouge_data< long double >::_S_cheby` [static]

Initial value:

```
{
    1.681473171108908244e+10L,
    -1.269150315503303974e+11L,
    4.339449429013039995e+11L,
    -8.893680202692714895e+11L,
    1.218472425867950986e+12L,
    -1.178403473259353616e+12L,
    8.282455311246278274e+11L,
    -4.292112878930625978e+11L,
    1.646988347276488710e+11L,
    -4.661514921989111004e+10L,
    9.619972564515443397e+09L,
    -1.419382551781042824e+09L,
    1.454145470816386107e+08L,
    -9.923020719435758179e+06L,
    4.253557563919127284e+05L,
    -1.053371059784341875e+04L,
    1.332425479537961437e+02L,
    -7.118343974029489132e-01L,
    1.172051640057979518e-03L,
    -3.323940885824119041e-07L,
    4.503801674404338524e-12L,
    -5.320477002211632680e-20L,
}
```

Definition at line 1821 of file `sf_gamma.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf_gamma.tcc](#)

10.25 `std::__detail::_Airy< _Tp >` Class Template Reference

Public Types

- using `scalar_type` = `std::__detail::__num_traits_t< value_type >`
- using `value_type` = `_Tp`

Public Member Functions

- `constexpr _Airy ()=default`
- `_Airy (const _Airy &)=default`
- `_Airy (_Airy &&)=default`
- `constexpr _AiryState< value_type > operator() (value_type __y) const`

Public Attributes

- `scalar_type inner_radius` { `_Airy_default_radII<scalar_type>::inner_radius`}
- `scalar_type outer_radius` { `_Airy_default_radII<scalar_type>::outer_radius`}

10.25.1 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 2497 of file sf_airy.tcc.

10.25.2 Member Typedef Documentation

10.25.2.1 `template<typename _Tp> using std::__detail::_Airy<_Tp>::scalar_type = std::__detail::__num_traits_↵
t<value_type>`

Definition at line 2502 of file sf_airy.tcc.

10.25.2.2 `template<typename _Tp> using std::__detail::_Airy<_Tp>::value_type = _Tp`

Definition at line 2501 of file sf_airy.tcc.

10.25.3 Constructor & Destructor Documentation

10.25.3.1 `template<typename _Tp> constexpr std::__detail::_Airy<_Tp>::_Airy() [default]`

10.25.3.2 `template<typename _Tp> std::__detail::_Airy<_Tp>::_Airy(const _Airy<_Tp> &) [default]`

10.25.3.3 `template<typename _Tp> std::__detail::_Airy<_Tp>::_Airy(_Airy<_Tp> &&) [default]`

10.25.4 Member Function Documentation

10.25.4.1 `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 2520 of file sf_airy.tcc.

References `std::__detail::__beta()`, `std::__detail::_Airy_series<_Tp>::_S_Ai()`, and `std::__detail::_Airy_series<_Tp>::_S_Bi()`.

10.25.5 Member Data Documentation

10.25.5.1 `template<typename _Tp> scalar_type std::__detail::_Airy< _Tp >::inner_radius`
`{ _Airy_default_radii<scalar_type>::inner_radius }`

Definition at line 2511 of file `sf_airy.tcc`.

10.25.5.2 `template<typename _Tp> scalar_type std::__detail::_Airy< _Tp >::outer_radius`
`{ _Airy_default_radii<scalar_type>::outer_radius }`

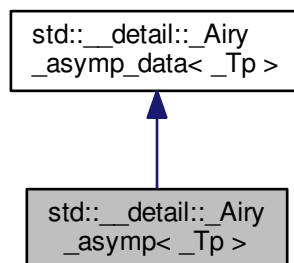
Definition at line 2512 of file `sf_airy.tcc`.

The documentation for this class was generated from the following file:

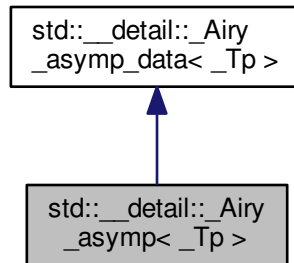
- [bits/sf_airy.tcc](#)

10.26 `std::__detail::_Airy_asymp< _Tp >` Class Template Reference

Inheritance diagram for `std::__detail::_Airy_asymp< _Tp >`:



Collaboration diagram for std::__detail::_Airy_asymp<_Tp>:



Public Types

- using `_Cmplx` = `std::complex<_Tp>`

Public Member Functions

- `constexpr _Airy_asymp()` = default
- `_AiryState<_Cmplx> _S_absarg_ge_pio3(_Cmplx __z) const`
*This function evaluates $Ai(z)$, $Ai'(z)$ and $Bi(z)$, $Bi'(z)$ from their asymptotic expansions for $|\arg(z)| < 2 * \pi/3$ i.e. roughly along the negative real axis.*
- `_AiryState<_Cmplx> _S_absarg_lt_pio3(_Cmplx __z) const`
This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|\arg(-z)| < \pi/3$ i.e. roughly along the negative real axis.
- `_AiryState<_Cmplx> operator()(_Cmplx __t, bool __return_fock_airy=false) const`

10.26.1 Detailed Description

```
template<typename _Tp>
class std::__detail::_Airy_asymp<_Tp>
```

A class encapsulating the asymptotic expansions of Airy functions and their derivatives.

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 1998 of file `sf_airy.tcc`.

10.26.2 Member Typedef Documentation

10.26.2.1 `template<typename _Tp> using std::__detail::_Airy_asymp< _Tp>::_Cmplx = std::complex<_Tp>`

Definition at line 2003 of file sf_airy.tcc.

10.26.3 Constructor & Destructor Documentation

10.26.3.1 `template<typename _Tp> constexpr std::__detail::_Airy_asymp< _Tp>::_Airy_asymp() [default]`

10.26.4 Member Function Documentation

10.26.4.1 `template<typename _Tp> _AiryState< std::complex< _Tp> > > std::__detail::_Airy_asymp< _Tp>::_S_absarg_ge_pio3(_Cmplx __z) const`

This function evaluates $Ai(z)$, $Ai'(z)$ and $Bi(z)$, $Bi'(z)$ from their asymptotic expansions for $|arg(z)| < 2 * \pi/3$ i.e. roughly along the negative real axis.

Template Parameters

<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 2271 of file sf_airy.tcc.

References `std::__detail::_AiryState< _Tp>::_z`.

10.26.4.2 `template<typename _Tp> _AiryState< std::complex< _Tp> > > std::__detail::_Airy_asymp< _Tp>::_S_absarg_lt_pio3(_Cmplx __z) const`

This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(-z)| < \pi/3$ i.e. roughly along the negative real axis.

For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined for $|z|$. This function assumes $|z| > 15$ and $|arg(-z)| < \pi/3$.

Note that for speed and since this function is called by another, checks for valid arguments are not made. Hence, an error return is not needed.

Template Parameters

<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 2301 of file sf_airy.tcc.

References std::__detail::_AiryState<_Tp>::__z.

10.26.4.3 `template<typename _Tp> _AiryState< std::complex<_Tp>> std::__detail::_Airy_asymp<_Tp>::operator()
(_Cmplx __t, bool __return_fock_airy = false) const`

Return the Airy functions for a given argument using asymptotic series.

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 2029 of file sf_airy.tcc.

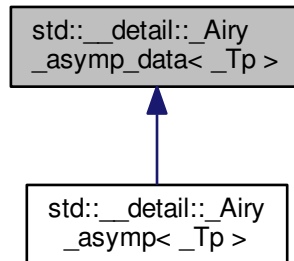
References std::__detail::_AiryState<_Tp>::__z.

The documentation for this class was generated from the following file:

- [bits/sf_airy.tcc](#)

10.27 std::__detail::_Airy_asymp_data< _Tp > Struct Template Reference

Inheritance diagram for std::__detail::_Airy_asymp_data< _Tp >:



10.27.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_Airy_asymp_data< _Tp >
```

A class encapsulating data for the asymptotic expansions of Airy functions and their derivatives.

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 632 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf_airy.tcc](#)

10.28 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

10.28.1 Detailed Description

```
template<>
struct std::__detail::_Airy_asymp_data< double >
```

Definition at line 739 of file sf_airy.tcc.

10.28.2 Member Data Documentation

10.28.2.1 constexpr double std::__detail::_Airy_asymp_data< double >::_S_c[_S_max_cd] [static]

Definition at line 745 of file sf_airy.tcc.

10.28.2.2 constexpr double std::__detail::_Airy_asymp_data< double >::_S_d[_S_max_cd] [static]

Definition at line 948 of file sf_airy.tcc.

10.28.2.3 constexpr int std::__detail::_Airy_asymp_data< double >::_S_max_cd = 198 [static]

Definition at line 741 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- [bits/sf_airy.tcc](#)

10.29 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

10.29.1 Detailed Description

```
template<>
struct std::__detail::_Airy_asymp_data< float >
```

Definition at line 636 of file sf_airy.tcc.

10.29.2 Member Data Documentation

10.29.2.1 `constexpr float std::__detail::_Airy_asymp_data< float >::_S_c[_S_max_cd]` `[static]`

Definition at line 642 of file `sf_airy.tcc`.

10.29.2.2 `constexpr float std::__detail::_Airy_asymp_data< float >::_S_d[_S_max_cd]` `[static]`

Definition at line 690 of file `sf_airy.tcc`.

10.29.2.3 `constexpr int std::__detail::_Airy_asymp_data< float >::_S_max_cd = 43` `[static]`

Definition at line 638 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf_airy.tcc](#)

10.30 `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`

10.30.1 Detailed Description

```
template<>
struct std::__detail::_Airy_asymp_data< long double >
```

Definition at line 1152 of file `sf_airy.tcc`.

10.30.2 Member Data Documentation

10.30.2.1 `constexpr long double std::__detail::_Airy_asymp_data< long double >::_S_c[_S_max_cd]` `[static]`

Definition at line 1158 of file `sf_airy.tcc`.

10.30.2.2 constexpr long double std::__detail::_Airy_asymp_data< long double >::_S_d[_S_max_cd] [static]

Definition at line 1364 of file sf_airy.tcc.

10.30.2.3 constexpr int std::__detail::_Airy_asymp_data< long double >::_S_max_cd = 201 [static]

Definition at line 1154 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- bits/sf_airy.tcc

10.31 std::__detail::_Airy_asymp_series< _Sum > Class Template Reference

Public Types

- using [scalar_type](#) = std::__detail::_num_traits_t< [value_type](#) >
- using [value_type](#) = typename _Sum::value_type

Public Member Functions

- [_Airy_asymp_series](#) (_Sum __proto)
- [_Airy_asymp_series](#) (const [_Airy_asymp_series](#) &)=default
- [_Airy_asymp_series](#) ([_Airy_asymp_series](#) &&)=default
- [_AiryState](#)< [value_type](#) > [operator\(\)](#) ([value_type](#) __y)

Static Public Attributes

- static constexpr [scalar_type](#) [_S_sqrt_pi](#) = __gnu_cxx::__const_root_pi([scalar_type](#){})

10.31.1 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 2364 of file sf_airy.tcc.

10.31.2 Member Typedef Documentation

10.31.2.1 `template<typename _Sum> using std::__detail::_Airy_asymp_series< _Sum >::scalar_type = std::__detail::_num_traits_t<value_type>`

Definition at line 2369 of file sf_airy.tcc.

10.31.2.2 `template<typename _Sum> using std::__detail::_Airy_asymp_series< _Sum >::value_type = typename _Sum::value_type`

Definition at line 2368 of file sf_airy.tcc.

10.31.3 Constructor & Destructor Documentation

10.31.3.1 `template<typename _Sum> std::__detail::_Airy_asymp_series< _Sum >::_Airy_asymp_series (_Sum __proto) [inline]`

Definition at line 2373 of file sf_airy.tcc.

10.31.3.2 `template<typename _Sum> std::__detail::_Airy_asymp_series< _Sum >::_Airy_asymp_series (const _Airy_asymp_series< _Sum > &) [default]`

10.31.3.3 `template<typename _Sum> std::__detail::_Airy_asymp_series< _Sum >::_Airy_asymp_series (_Airy_asymp_series< _Sum > &&) [default]`

10.31.4 Member Function Documentation

10.31.4.1 `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 2418 of file sf_airy.tcc.

10.31.5 Member Data Documentation

10.31.5.1 `template<typename _Sum> constexpr _Airy_asymp_series< _Sum >::scalar_type
std::__detail::_Airy_asymp_series< _Sum >::S_sqrt_pi = __gnu_cxx::__const_root_pi(scalar_type{})
[static]`

Definition at line 2371 of file sf_airy.tcc.

The documentation for this class was generated from the following file:

- [bits/sf_airy.tcc](#)

10.32 std::__detail::_Airy_default_radii< _Tp > Struct Template Reference

10.32.1 Detailed Description

```
template<typename _Tp>  
struct std::__detail::_Airy_default_radii< _Tp >
```

Definition at line 2468 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- [bits/sf_airy.tcc](#)

10.33 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}

10.33.1 Detailed Description

```
template<>  
struct std::__detail::_Airy_default_radii< double >
```

Definition at line 2479 of file sf_airy.tcc.

10.33.2 Member Data Documentation

10.33.2.1 `constexpr double std::__detail::_Airy_default_radii< double >::inner_radius {4.0} [static]`

Definition at line 2481 of file `sf_airy.tcc`.

10.33.2.2 `constexpr double std::__detail::_Airy_default_radii< double >::outer_radius {12.0} [static]`

Definition at line 2482 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf_airy.tcc](#)

10.34 `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}

10.34.1 Detailed Description

```
template<>
struct std::__detail::_Airy_default_radii< float >
```

Definition at line 2472 of file `sf_airy.tcc`.

10.34.2 Member Data Documentation

10.34.2.1 `constexpr float std::__detail::_Airy_default_radii< float >::inner_radius {2.0F} [static]`

Definition at line 2474 of file `sf_airy.tcc`.

10.34.2.2 `constexpr float std::__detail::_Airy_default_radii< float >::outer_radius {6.0F} [static]`

Definition at line 2475 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- [bits/sf_airy.tcc](#)

10.35 `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}

10.35.1 Detailed Description

```
template<>
struct std::__detail::_Airy_default_radii< long double >
```

Definition at line 2486 of file `sf_airy.tcc`.

10.35.2 Member Data Documentation

10.35.2.1 constexpr long double `std::__detail::_Airy_default_radii< long double >::inner_radius` {5.0L} `[static]`

Definition at line 2488 of file `sf_airy.tcc`.

10.35.2.2 constexpr long double `std::__detail::_Airy_default_radii< long double >::outer_radius` {15.0L} `[static]`

Definition at line 2489 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- `bits/sf_airy.tcc`

10.36 `std::__detail::_Airy_series< _Tp >` Class Template Reference

Public Types

- using [_Cmplx](#) = `std::complex< _Tp >`

Static Public Member Functions

- static `std::pair< _Cmplx, _Cmplx > _S_Ai` ([_Cmplx](#) __t)
- static `_AiryState< _Cmplx > _S_Airy` ([_Cmplx](#) __t)
- static `std::pair< _Cmplx, _Cmplx > _S_Bi` ([_Cmplx](#) __t)
- static `_AiryAuxilliaryState< _Cmplx > _S_FGH` ([_Cmplx](#) __t)
- static `_AiryState< _Cmplx > _S_Fock` ([_Cmplx](#) __t)
- static `_AiryState< _Cmplx > _S_Scorer` ([_Cmplx](#) __t)
- static `_AiryState< _Cmplx > _S_Scorer2` ([_Cmplx](#) __t)

Static Public Attributes

- static constexpr int `_N_FGH` = 200
- static constexpr `_Tp` `_S_Ai0` = `_Tp{3.550280538878172392600631860041831763980e-1L}`
- static constexpr `_Tp` `_S_Aip0` = `_Tp{-2.588194037928067984051835601892039634793e-1L}`
- static constexpr `_Tp` `_S_Bi0` = `_Tp{6.149266274460007351509223690936135535960e-1L}`
- static constexpr `_Tp` `_S_Bip0` = `_Tp{4.482883573538263579148237103988283908668e-1L}`
- static constexpr `_Tp` `_S_eps` = `__gnu_cxx::__epsilon(_Tp{})`
- static constexpr `_Tp` `_S_Gi0` = `_Tp{2.049755424820002450503074563645378511979e-1L}`
- static constexpr `_Tp` `_S_Gip0` = `_Tp{1.494294524512754526382745701329427969551e-1L}`
- static constexpr `_Tp` `_S_Hi0` = `_Tp{4.099510849640004901006149127290757023959e-1L}`
- static constexpr `_Tp` `_S_Hip0` = `_Tp{2.988589049025509052765491402658855939102e-1L}`
- static constexpr `_Cmplx_S_i` {`_Tp{0}`, `_Tp{1}`}
- static constexpr `_Tp` `_S_pi` = `__gnu_cxx::__const_pi(_Tp{})`
- static constexpr `_Tp` `_S_sqrt_pi` = `__gnu_cxx::__const_root_pi(_Tp{})`

10.36.1 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`.

10.36.2 Member Typedef Documentation

10.36.2.1 `template<typename _Tp> using std::__detail::_Airy_series< _Tp >::_Cmplx = std::complex<_Tp>`

Definition at line 112 of file `sf_airy.tcc`.

10.36.3 Member Function Documentation

10.36.3.1 `template<typename _Tp> std::pair< std::complex<_Tp>, std::complex<_Tp> > std::__detail::_Airy_series<_Tp>::S_Ai(_Cmplx __t) [static]`

Return the Airy function of the first kind and its derivative by using the series expansions of the auxilliary Airy functions:

$$f_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$g_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

The Airy function of the first kind is then defined by:

$$Ai(x) = Ai(0)f_{ai}(x) + Ai'(0)g_{ai}(x)$$

where $Ai(0) = 3^{-2/3}/\Gamma(2/3)$, $Ai'(0) = -3-1/2Bi'(0)$ 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 341 of file sf_airy.tcc.

Referenced by `std::__detail::_Airy<_Tp>::operator()()`.

10.36.3.2 `template<typename _Tp> _AiryState< std::complex<_Tp> > std::__detail::_Airy_series<_Tp>::S_Airy(_Cmplx __t) [static]`

Return the Fock-type Airy functions $Ai(t)$, and $Bi(t)$ and their derivatives of complex argument.

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Parameters

<code>↵</code>	The complex argument
<code>__↵</code>	
<code>↵</code>	
<code>__↵</code>	
<code>t</code>	

Definition at line 609 of file sf_airy.tcc.

10.36.3.3 `template<typename _Tp> std::pair< std::complex< _Tp>, std::complex< _Tp> > std::__detail::_Airy_series< _Tp>::_S_Bi(_Cmplx __t) [static]`

Return the Airy function of the second kind and its derivative by using the series expansions of the auxilliary Airy functions:

$$f_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$g_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

The Airy function of the second kind is then defined by:

$$Bi(x) = Bi(0)f_{ai}(x) + Bi'(0)g_{ai}(x)$$

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 364 of file `sf_airy.tcc`.

Referenced by `std::__detail::_Airy< _Tp>::operator()()`.

10.36.3.4 `template<typename _Tp> _AiryAuxilliaryState< std::complex< _Tp> > std::__detail::_Airy_series< _Tp>::_S_FGH(_Cmplx __t) [static]`

Return the auxilliary Airy functions:

$$f_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$g_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$h_{ai}(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 383 of file `sf_airy.tcc`.

10.36.3.5 `template<typename _Tp> _AiryState< std::complex< _Tp> > std::__detail::_Airy_series< _Tp>::_S_Fock(_Cmplx __t) [static]`

Return the Fock-type Airy functions $w_1(t)$, and $w_2(t)$ and their derivatives of complex argument.

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Parameters

<code>↵</code>	The complex argument
<code>↵</code>	
<code>↵</code>	
<code>↵</code>	
<code>t</code>	

Definition at line 621 of file sf_airy.tcc.

10.36.3.6 `template<typename _Tp> _AiryState< std::complex<_Tp>> std::__detail::_Airy_series<_Tp>::_S_Scorer (_Cmplx_t) [static]`

Return the Scorer functions by using the series expansions of the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$hai(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

The Scorer function is then defined by:

$$Hi(x) = Hi(0) (fai(x) + gai(x) + hai(x))$$

where $Hi(0) = 2/(3^{7/6}\Gamma(2/3))$ and $Hi'(0) = 2/(3^{5/6}\Gamma(1/3))$. The other Scorer function is found from the identity

$$Gi(x) + Hi(x) = Bi(x)$$

Todo Find out what is wrong with the $Hi = fai + gai + hai$ scorer function.

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 464 of file sf_airy.tcc.

```
10.36.3.7 template<typename _Tp > _AiryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp
>::_S_Scorer2( _Cmplx_t ) [static]
```

Return the Scorer functions by using the series expansions:

$$\begin{aligned}
 Hi(x) &= \frac{3^{-2/3}}{\pi} \sum_{k=0}^{\infty} \Gamma\left(\frac{k+1}{3}\right) \frac{3^{1/3}x}{k!} \\
 Hi'(x) &= \frac{3^{-1/3}}{\pi} \sum_{k=0}^{\infty} \Gamma\left(\frac{k+2}{3}\right) \frac{3^{1/3}x}{k!} \\
 Gi(x) &= \frac{3^{-2/3}}{\pi} \sum_{k=0}^{\infty} \cos\left(\frac{2k-1}{3}\pi\right) \Gamma\left(\frac{k+1}{3}\right) \frac{3^{1/3}x}{k!} \\
 Gi'(x) &= \frac{3^{-1/3}}{\pi} \sum_{k=0}^{\infty} \cos\left(\frac{2k+1}{3}\pi\right) \Gamma\left(\frac{k+2}{3}\right) \frac{3^{1/3}x}{k!}
 \end{aligned}$$

Definition at line 501 of file sf_airy.tcc.

References std::__detail::__gamma().

10.36.4 Member Data Documentation

```
10.36.4.1 template<typename _Tp > constexpr int std::__detail::_Airy_series< _Tp >::_N_FGH = 200 [static]
```

Definition at line 114 of file sf_airy.tcc.

```
10.36.4.2 template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Ai0 =
_Tp{3.550280538878172392600631860041831763980e-1L} [static]
```

Definition at line 130 of file sf_airy.tcc.

```
10.36.4.3 template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Aip0 =
_Tp{-2.588194037928067984051835601892039634793e-1L} [static]
```

Definition at line 132 of file sf_airy.tcc.

```
10.36.4.4 template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Bi0 =
_Tp{6.149266274460007351509223690936135535960e-1L} [static]
```

Definition at line 134 of file sf_airy.tcc.

10.36.4.5 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Bip0 =
_Tp{4.482883573538263579148237103988283908668e-1L} [static]`

Definition at line 136 of file sf_airy.tcc.

10.36.4.6 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_eps =
__gnu_cxx::__epsilon(_Tp{}) [static]`

Definition at line 125 of file sf_airy.tcc.

10.36.4.7 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Gi0 =
_Tp{2.049755424820002450503074563645378511979e-1L} [static]`

Definition at line 142 of file sf_airy.tcc.

10.36.4.8 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Gip0 =
_Tp{1.494294524512754526382745701329427969551e-1L} [static]`

Definition at line 144 of file sf_airy.tcc.

10.36.4.9 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Hi0 =
_Tp{4.099510849640004901006149127290757023959e-1L} [static]`

Definition at line 138 of file sf_airy.tcc.

10.36.4.10 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Hip0 =
_Tp{2.988589049025509052765491402658855939102e-1L} [static]`

Definition at line 140 of file sf_airy.tcc.

10.36.4.11 `template<typename _Tp> constexpr std::complex<_Tp> std::__detail::_Airy_series<_Tp>::_S_i {_Tp{0},
_Tp{1}} [static]`

Definition at line 145 of file sf_airy.tcc.

10.36.4.12 `template<typename _Tp> constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_pi =
__gnu_cxx::__const_pi(_Tp{}) [static]`

Definition at line 126 of file sf_airy.tcc.

```
10.36.4.13 template<typename _Tp > constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_sqrt_pi =
    __gnu_cxx::__const_root_pi(_Tp{}) [static]
```

Definition at line 128 of file sf_airy.tcc.

The documentation for this class was generated from the following file:

- bits/sf_airy.tcc

10.37 std::__detail::_AiryAuxilliaryState< _Tp > Struct Template Reference

Public Types

- using [_Val](#) = std::__detail::__num_traits_t< _Tp >

Public Attributes

- [_Tp __fai_deriv](#)
- [_Tp __fai_value](#)
- [_Tp __gai_deriv](#)
- [_Tp __gai_value](#)
- [_Tp __hai_deriv](#)
- [_Tp __hai_value](#)
- [_Tp __z](#)

10.37.1 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.

10.37.2 Member Typedef Documentation

```
10.37.2.1 template<typename _Tp> using std::__detail::_AiryAuxilliaryState< _Tp >::_Val =
    std::__detail::__num_traits_t< _Tp>
```

Definition at line 82 of file sf_airy.tcc.

10.37.3 Member Data Documentation

10.37.3.1 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::__fai_deriv`

Definition at line 86 of file `sf_airy.tcc`.

10.37.3.2 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::__fai_value`

Definition at line 85 of file `sf_airy.tcc`.

10.37.3.3 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::__gai_deriv`

Definition at line 88 of file `sf_airy.tcc`.

10.37.3.4 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::__gai_value`

Definition at line 87 of file `sf_airy.tcc`.

10.37.3.5 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::__hai_deriv`

Definition at line 90 of file `sf_airy.tcc`.

10.37.3.6 `template<typename _Tp> _Tp std::__detail::_AiryAuxilliaryState< _Tp >::__hai_value`

Definition at line 89 of file `sf_airy.tcc`.

10.37.3.7 `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](#)

10.38 std::__detail::_AiryState< _Tp > Struct Template Reference

Public Types

- using `_Real` = `std::__detail::__num_traits_t< _Tp >`

Public Member Functions

- [_Real true_Wronskian](#) ()
- [_Tp Wronskian](#) () const

Public Attributes

- [_Tp __Ai_deriv](#)
- [_Tp __Ai_value](#)
- [_Tp __Bi_deriv](#)
- [_Tp __Bi_value](#)
- [_Tp __z](#)

10.38.1 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.

10.38.2 Member Typedef Documentation

10.38.2.1 `template<typename _Tp> using std::__detail::_AiryState< _Tp >::_Real = std::__detail::__num_traits_t<_Tp>`

Definition at line 57 of file sf_airy.tcc.

10.38.3 Member Function Documentation

10.38.3.1 `template<typename _Tp> _Real std::__detail::_AiryState< _Tp >::true_Wronskian () [inline]`

Definition at line 70 of file sf_airy.tcc.

10.38.3.2 `template<typename _Tp> _Tp std::__detail::_AiryState< _Tp >::Wronskian () const [inline]`

Definition at line 66 of file sf_airy.tcc.

References `std::__detail::_AiryState< _Tp >::__Ai_deriv`.

10.38.4 Member Data Documentation

10.38.4.1 `template<typename _Tp> _Tp std::__detail::_AiryState<_Tp>::__Ai_deriv`

Definition at line 61 of file `sf_airy.tcc`.

Referenced by `std::__detail::_AiryState<_Tp>::Wronskian()`.

10.38.4.2 `template<typename _Tp> _Tp std::__detail::_AiryState<_Tp>::__Ai_value`

Definition at line 60 of file `sf_airy.tcc`.

10.38.4.3 `template<typename _Tp> _Tp std::__detail::_AiryState<_Tp>::__Bi_deriv`

Definition at line 63 of file `sf_airy.tcc`.

10.38.4.4 `template<typename _Tp> _Tp std::__detail::_AiryState<_Tp>::__Bi_value`

Definition at line 62 of file `sf_airy.tcc`.

10.38.4.5 `template<typename _Tp> _Tp std::__detail::_AiryState<_Tp>::__z`

Definition at line 59 of file `sf_airy.tcc`.

Referenced by `std::__detail::_Airy_asymp<_Tp>::S_absarg_ge_pio3()`, `std::__detail::_Airy_asymp<_Tp>::S_↵absarg_lt_pio3()`, and `std::__detail::_Airy_asymp<_Tp>::operator()()`.

The documentation for this struct was generated from the following file:

- [bits/sf_airy.tcc](#)

10.39 std::__detail::_AsympTerminator<_Tp> Class Template Reference

Public Member Functions

- [_AsympTerminator](#) (`std::size_t __max_iter, _Real __mul=_Real{1}`)
- [operator\(\)](#) (`_Tp __term, _Tp __sum`)

10.39.1 Detailed Description

```
template<typename _Tp>
class std::__detail::_AsympTerminator< _Tp >
```

This class manages the termination of asymptotic series. In particular, this termination watches for the growth of the sequence of terms to stop the series.

Termination conditions involve both a maximum iteration count and a relative precision.

Definition at line 99 of file sf_polylog.tcc.

10.39.2 Constructor & Destructor Documentation

10.39.2.1 `template<typename _Tp> std::__detail::_AsympTerminator< _Tp >::_AsympTerminator (std::size_t __max_iter, _Real __mul = _Real{1}) [inline]`

Definition at line 110 of file sf_polylog.tcc.

10.39.3 Member Function Documentation

10.39.3.1 `template<typename _Tp> bool std::__detail::_AsympTerminator< _Tp >::operator() (_Tp __term, _Tp __sum) [inline]`

Definition at line 116 of file sf_polylog.tcc.

The documentation for this class was generated from the following file:

- [bits/sf_polylog.tcc](#)

10.40 std::__detail::_Factorial_table< _Tp > Struct Template Reference

Public Attributes

- [_Tp __factorial](#)
- [_Tp __log_factorial](#)
- [int __n](#)

10.40.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_Factorial_table< _Tp >
```

Definition at line 65 of file sf_gamma.tcc.

10.40.2 Member Data Documentation

10.40.2.1 template<typename _Tp> _Tp std::__detail::_Factorial_table< _Tp >::__factorial

Definition at line 68 of file sf_gamma.tcc.

Referenced by std::__detail::__double_factorial(), and std::__detail::__gamma_reciprocal().

10.40.2.2 template<typename _Tp> _Tp std::__detail::_Factorial_table< _Tp >::__log_factorial

Definition at line 69 of file sf_gamma.tcc.

Referenced by std::__detail::__log_double_factorial(), and std::__detail::__log_gamma().

10.40.2.3 template<typename _Tp> int std::__detail::_Factorial_table< _Tp >::__n

Definition at line 67 of file sf_gamma.tcc.

Referenced by std::__detail::__binomial(), std::__detail::__double_factorial(), std::__detail::__factorial(), std::__detail::__falling_factorial(), std::__detail::__gamma(), std::__detail::__gamma_cont_frac(), std::__detail::__gamma_reciprocal(), std::__detail::__gamma_series(), std::__detail::__harmonic_number(), std::__detail::__lanczos_binet1p(), std::__detail::__log_binomial(), std::__detail::__log_binomial_sign(), std::__detail::__log_double_factorial(), std::__detail::__log_factorial(), std::__detail::__log_gamma(), std::__detail::__psi(), and std::__detail::__rising_factorial().

The documentation for this struct was generated from the following file:

- bits/sf_gamma.tcc

10.41 std::__detail::_Terminator< _Tp > Class Template Reference

Public Member Functions

- [_Terminator](#) (std::size_t __max_iter, _Real __mul=_Real{1})
- bool [operator\(\)](#) (_Tp __term, _Tp __sum)

10.41.1 Detailed Description

```
template<typename _Tp>
class std::__detail::_Terminator< _Tp >
```

This class manages the termination of series. Termination conditions involve both a maximum iteration count and a relative precision.

Definition at line 63 of file sf_polylog.tcc.

10.41.2 Constructor & Destructor Documentation

10.41.2.1 `template<typename _Tp> std::__detail::_Terminator<_Tp>::_Terminator (std::size_t __max_iter, _Real __mul =_Real{1}) [inline]`

Definition at line 73 of file `sf_polylog.tcc`.

10.41.3 Member Function Documentation

10.41.3.1 `template<typename _Tp> bool std::__detail::_Terminator<_Tp>::operator() (_Tp __term, _Tp __sum) [inline]`

Definition at line 79 of file `sf_polylog.tcc`.

The documentation for this class was generated from the following file:

- [bits/sf_polylog.tcc](#)

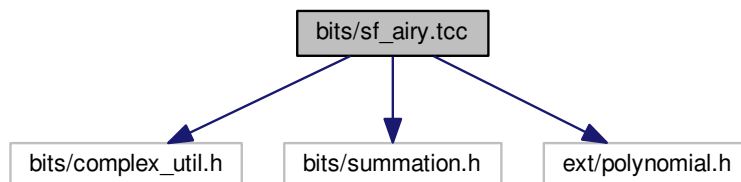
Chapter 11

File Documentation

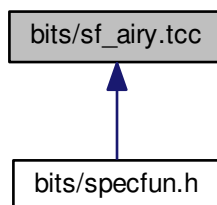
11.1 bits/sf_airy.tcc File Reference

```
#include <bits/complex_util.h>
#include <bits/summation.h>
#include <ext/polynomial.h>
```

Include dependency graph for sf_airy.tcc:



This graph shows which files directly or indirectly include this file:



Classes

- class `std::__detail::_Airy<_Tp>`
- class `std::__detail::_Airy_asymp<_Tp>`
- struct `std::__detail::_Airy_asymp_data<_Tp>`
- struct `std::__detail::_Airy_asymp_data<double>`
- struct `std::__detail::_Airy_asymp_data<float>`
- struct `std::__detail::_Airy_asymp_data<long double>`
- class `std::__detail::_Airy_asymp_series<_Sum>`
- struct `std::__detail::_Airy_default_radii<_Tp>`
- struct `std::__detail::_Airy_default_radii<double>`
- struct `std::__detail::_Airy_default_radii<float>`
- struct `std::__detail::_Airy_default_radii<long double>`
- class `std::__detail::_Airy_series<_Tp>`
- struct `std::__detail::_AiryAuxilliaryState<_Tp>`
- struct `std::__detail::_AiryState<_Tp>`

Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_BITS_SF_AIRY_TCC 1`

Functions

- `template<typename _Tp>`
`std::complex<_Tp> std::__detail::__airy_ai (std::complex<_Tp> __z)`
Return the complex Airy Ai function.
- `template<typename _Tp>`
`std::complex<_Tp> std::__detail::__airy_bi (std::complex<_Tp> __z)`
Return the complex Airy Bi function.

Variables

- `template<typename _Tp>`
`constexpr int std::__detail::__max_FGH = _Airy_series<_Tp>::_N_FGH`
- `template<>`
`constexpr int std::__detail::__max_FGH<double> = 79`
- `template<>`
`constexpr int std::__detail::__max_FGH<float> = 15`

11.1.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

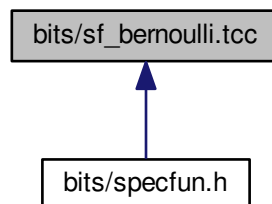
11.1.2 Macro Definition Documentation

11.1.2.1 `#define _GLIBCXX_BITS_SF_AIRY_TCC 1`

Definition at line 31 of file sf_airy.tcc.

11.2 bits/sf_bernoulli.tcc File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_BERNOULLI_TCC 1](#)

Functions

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (unsigned int __n)`
This returns Bernoulli number B_n .
- `template<typename _Tp >`
`_Tp std::__detail::__bernoulli (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (unsigned int __n)`
This returns Bernoulli number B_{2n} at even integer arguments $2n$.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series (unsigned int __n)`
This returns Bernoulli numbers from a table or by summation for larger values.

$$B_{2n} = (-1)^{n+1} 2 \frac{(2n)!}{(2\pi)^{2n}} \zeta(2n)$$

11.2.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

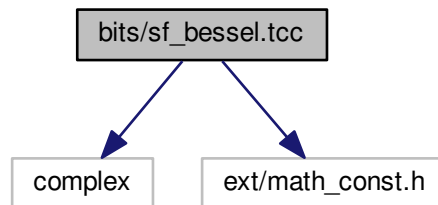
11.2.2 Macro Definition Documentation

11.2.2.1 `#define _GLIBCXX_BITS_SF_BERNOULLI_TCC 1`

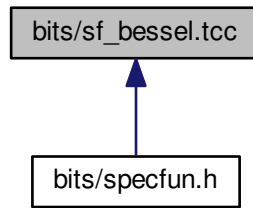
Definition at line 35 of file `sf_bernoulli.tcc`.

11.3 `bits/sf_bessel.tcc` File Reference

```
#include <complex>
#include <ext/math_const.h>
Include dependency graph for sf_bessel.tcc:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_BESSEL_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)`
This routine returns the cylindrical Bessel functions of order ν : J_ν or I_ν by series expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__cyl_bessel_j (_Tp __nu, _Tp __x)`
Return the Bessel function of order ν : $J_\nu(x)$.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn (_Tp __nu, _Tp __x)`
Return the cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn_asymp (_Tp __nu, _Tp __x)`
This routine computes the asymptotic cylindrical Bessel and Neumann functions of order ν : $J_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, std::complex<_Tp>> std::__detail::__cyl_bessel_jn_neg_arg (_Tp \leftrightarrow __nu, _Tp __x)`
Return the cylindrical Bessel functions and their derivatives of order ν and argument $x < 0$.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn_steud (_Tp __nu, _Tp __x)`

Compute the Bessel $J_\nu(x)$ and Neumann $N_\nu(x)$ functions and their first derivatives $J'_\nu(x)$ and $N'_\nu(x)$ respectively. These four functions are computed together for numerical stability.

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_1 (_Tp __nu, _Tp __x)`
 Return the cylindrical Hankel function of the first kind $H_\nu^{(1)}(x)$.
- `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 >`
`_gnu_cxx::__gamma_temme_t< _Tp > std::__detail::__gamma_temme (_Tp __mu)`
 Compute the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ 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 >`
`_gnu_cxx::__sph_bessel_t< unsigned int, _Tp, _Tp > std::__detail::__sph_bessel_jn (unsigned int __n, _Tp __x)`
 Compute the spherical Bessel $j_n(x)$ and Neumann $n_n(x)$ functions and their first derivatives $j'_n(x)$ and $n'_n(x)$ respectively.
- `template<typename _Tp >`
`_gnu_cxx::__sph_bessel_t< unsigned int, _Tp, std::complex< _Tp > > std::__detail::__sph_bessel_jn_neg_arg (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, _Tp __x)`
 Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.
- `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 .

11.3.1 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

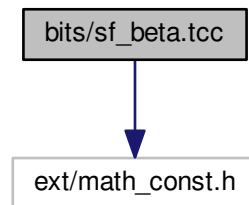
11.3.2.1 `#define _GLIBCXX_BITS_SF_BESSEL_TCC 1`

Definition at line 47 of file sf_bessel.tcc.

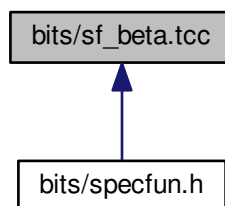
11.4 bits/sf_beta.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_beta.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- `#define _GLIBCXX_BITS_SF_BETA_TCC 1`

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__beta (_Tp __a, _Tp __b)`
Return the beta function $B(a, b)$.
- `template<typename _Tp >`
`_Tp std::__detail::__beta_gamma (_Tp __a, _Tp __b)`
Return the beta function: $B(a, b)$.
- `template<typename _Tp >`
`_Tp std::__detail::__beta_inc (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__beta_lgamma (_Tp __a, _Tp __b)`
Return the beta function $B(a, b)$ using the log gamma functions.
- `template<typename _Tp >`
`_Tp std::__detail::__beta_product (_Tp __a, _Tp __b)`
Return the beta function $B(x, y)$ using the product form.
- `template<typename _Tp >`
`_Tp std::__detail::__ibeta_cont_frac (_Tp __a, _Tp __b, _Tp __x)`

11.4.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.4.2 Macro Definition Documentation

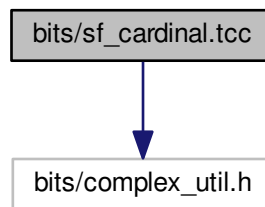
11.4.2.1 `#define _GLIBCXX_BITS_SF_BETA_TCC 1`

Definition at line 49 of file `sf_beta.tcc`.

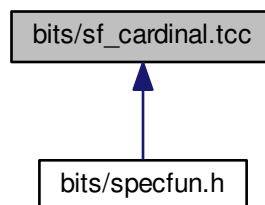
11.5 bits/sf_cardinal.tcc File Reference

```
#include <bits/complex_util.h>
```

Include dependency graph for sf_cardinal.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_CARDINAL_TCC](#) 1

Functions

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::__detail::__sinc (_Tp __x)`
Return the sinus cardinal function

$$\text{sinc}(x) = \frac{\sin(x)}{x}$$
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::__detail::__sinc_pi (_Tp __x)`
Return the reperiodized sinus cardinal function

$$\text{sinc}_{\pi}(x) = \frac{\sin(\pi x)}{\pi x}$$
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::__detail::__sinhc (_Tp __x)`
Return the hyperbolic sinus cardinal function

$$\text{sinhc}(x) = \frac{\sinh(x)}{x}$$
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::__detail::__sinhc_pi (_Tp __x)`
Return the reperiodized hyperbolic sinus cardinal function

$$\text{sinhc}_{\pi}(x) = \frac{\sinh(\pi x)}{\pi x}$$

11.5.1 Macro Definition Documentation

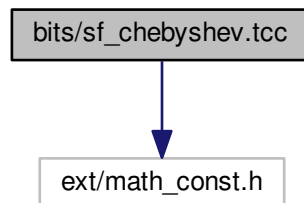
11.5.1.1 `#define _GLIBCXX_BITS_SF_CARDINAL_TCC 1`

Definition at line 31 of file `sf_cardinal.tcc`.

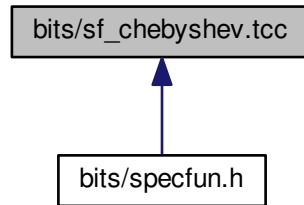
11.6 `bits/sf_chebyshev.tcc` File Reference

`#include <ext/math_const.h>`

Include dependency graph for `sf_chebyshev.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_CHEBYSHEV_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__chebyshev_recur \(unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1\)](#)
- [template<typename _Tp > _Tp std::__detail::__chebyshev_t \(unsigned int __n, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__chebyshev_u \(unsigned int __n, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__chebyshev_v \(unsigned int __n, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__chebyshev_w \(unsigned int __n, _Tp __x\)](#)

11.6.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.6.2 Macro Definition Documentation

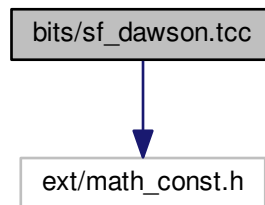
11.6.2.1 `#define _GLIBCXX_BITS_SF_CHEBYSHEV_TCC 1`

Definition at line 31 of file `sf_chebyshev.tcc`.

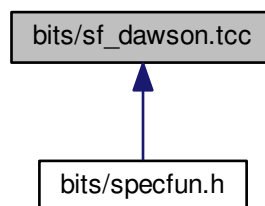
11.7 `bits/sf_dawson.tcc` File Reference

```
#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.

11.7.1 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

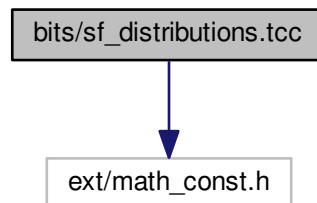
11.7.2.1 `#define _GLIBCXX_BITS_SF_DAWSON_TCC 1`

Definition at line 31 of file `sf_dawson.tcc`.

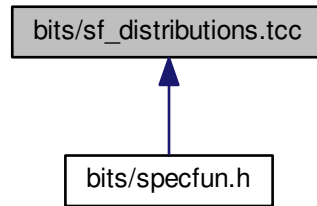
11.8 bits/sf_distributions.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for `sf_distributions.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- `#define GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC 1`

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)`
Return the binomial cumulative distribution function.
- `template<typename _Tp >`
`_Tp std::__detail::__binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)`
Return the complementary binomial cumulative distribution function.
- `template<typename _Tp >`
`_Tp std::__detail::__binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)`
Return the binomial probability mass function.
- `template<typename _Tp >`
`_Tp std::__detail::__chi_squared_pdf (_Tp __chi2, unsigned int __nu)`
Return the chi-squared propability function. This returns the probability that the observed chi-squared for a correct model is less than the value χ^2 .
- `template<typename _Tp >`
`_Tp std::__detail::__chi_squared_pdfc (_Tp __chi2, unsigned int __nu)`
Return the complementary chi-squared propability function. This returns the probability that the observed chi-squared for a correct model is greater than the value χ^2 .
- `template<typename _Tp >`
`_Tp std::__detail::__exponential_cdf (_Tp __lambda, _Tp __x)`
Return the exponential cumulative probability density function.

- `template<typename _Tp >`
`_Tp std::__detail::__exponential_cdf (_Tp __lambda, _Tp __x)`
Return the complement of the exponential cumulative probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__exponential_pdf (_Tp __lambda, _Tp __x)`
Return the exponential probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- `template<typename _Tp >`
`_Tp std::__detail::__fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)`
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- `template<typename _Tp >`
`_Tp std::__detail::__fisher_f_pdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- `template<typename _Tp >`
`_Tp std::__detail::__gamma_cdf (_Tp __alpha, _Tp __beta, _Tp __x)`
Return the gamma cumulative propability distribution function.
- `template<typename _Tp >`
`_Tp std::__detail::__gamma_cdfc (_Tp __alpha, _Tp __beta, _Tp __x)`
Return the gamma complementary cumulative propability distribution function.
- `template<typename _Tp >`
`_Tp std::__detail::__gamma_pdf (_Tp __alpha, _Tp __beta, _Tp __x)`
Return the gamma propability distribution function.
- `template<typename _Tp >`
`_Tp std::__detail::__logistic_cdf (_Tp __a, _Tp __b, _Tp __x)`
Return the logistic cumulative distribution function.
- `template<typename _Tp >`
`_Tp std::__detail::__logistic_pdf (_Tp __a, _Tp __b, _Tp __x)`
Return the logistic probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__lognormal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)`
Return the lognormal cumulative probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__lognormal_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`
Return the lognormal probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__normal_cdf (_Tp __mu, _Tp __sigma, _Tp __x)`
Return the normal cumulative probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__normal_pdf (_Tp __mu, _Tp __sigma, _Tp __x)`
Return the normal probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__rice_pdf (_Tp __nu, _Tp __sigma, _Tp __x)`
Return the Rice probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__student_t_cdf (_Tp __t, unsigned int __nu)`

Return the Students T probability function.

- `template<typename _Tp >`
`_Tp std::__detail::__student_t_cdfc (_Tp __t, unsigned int __nu)`

Return the complement of the Students T probability function.

- `template<typename _Tp >`
`_Tp std::__detail::__student_t_pdf (_Tp __t, unsigned int __nu)`

Return the Students T probability density.

- `template<typename _Tp >`
`_Tp std::__detail::__weibull_cdf (_Tp __a, _Tp __b, _Tp __x)`

Return the Weibull cumulative probability density function.

- `template<typename _Tp >`
`_Tp std::__detail::__weibull_pdf (_Tp __a, _Tp __b, _Tp __x)`

Return the Weibull probability density function.

11.8.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

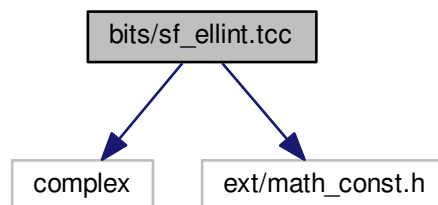
11.8.2 Macro Definition Documentation

11.8.2.1 `#define _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC 1`

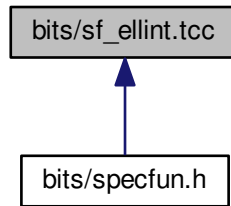
Definition at line 49 of file `sf_distributions.tcc`.

11.9 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)`

11.9.1 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

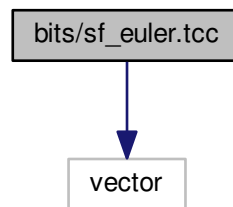
11.9.2.1 `#define GLIBCXX_BITS_SF_ELLINT_TCC 1`

Definition at line 47 of file `sf_ellint.tcc`.

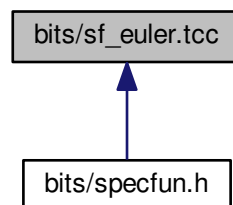
11.10 bits/sf_euler.tcc File Reference

```
#include <vector>
```

Include dependency graph for sf_euler.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_EULER_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__euler (unsigned int __n)`
This returns Euler number E_n .
- `template<typename _Tp >`
`_Tp std::__detail::__euler (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__euler_series (unsigned int __n)`
- `template<typename _Tp >`
`_Tp std::__detail::__eulerian_1_recur (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp std::__detail::__eulerian_2 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp std::__detail::__eulerian_2_recur (unsigned int __n, unsigned int __m)`

11.10.1 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

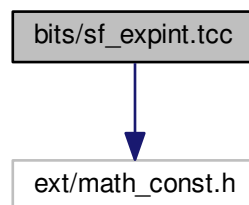
11.10.2.1 `#define _GLIBCXX_BITS_SF_EULER_TCC 1`

Definition at line 35 of file `sf_euler.tcc`.

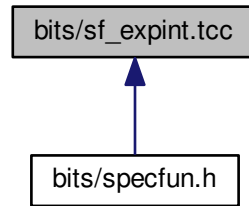
11.11 bits/sf_expint.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_expint.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- `#define _GLIBCXX_BITS_SF_EXPINT_TCC 1`

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__coshint (const _Tp __x)`
Return the hyperbolic cosine integral $Chi(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint (_Tp __x)`
Return the exponential integral $Ei(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_E1 (_Tp __x)`
Return the exponential integral $E_1(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_E1_asymp (_Tp __x)`
Return the exponential integral $E_1(x)$ by asymptotic expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_E1_series (_Tp __x)`
Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei (_Tp __x)`

Return the exponential integral $Ei(x)$.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei_asymp (_Tp __x)`

Return the exponential integral $Ei(x)$ by asymptotic expansion.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei_series (_Tp __x)`

Return the exponential integral $Ei(x)$ by series summation.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_asymp (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ for large argument.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_cont_frac (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by continued fractions.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_large_n (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ for large order.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_recursion (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_series (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by series summation.

- `template<typename _Tp >`
`_Tp std::__detail::__logint (const _Tp __x)`

Return the logarithmic integral $li(x)$.

- `template<typename _Tp >`
`_Tp std::__detail::__sinhint (const _Tp __x)`

Return the hyperbolic sine integral $Shi(x)$.

11.11.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

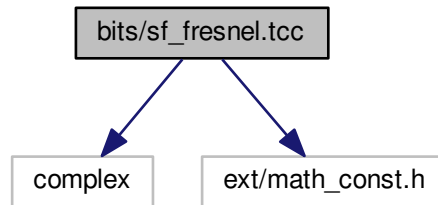
11.11.2 Macro Definition Documentation

11.11.2.1 `#define _GLIBCXX_BITS_SF_EXPINT_TCC 1`

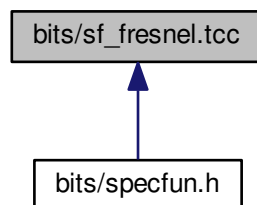
Definition at line 47 of file `sf_expint.tcc`.

11.12 bits/sf_fresnel.tcc File Reference

```
#include <complex>
#include <ext/math_const.h>
Include dependency graph for sf_fresnel.tcc:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_BITS_SF_FRESNEL_TCC 1`

Functions

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__fresnel (const _Tp __x)`
Return the Fresnel cosine and sine integrals as a complex number $C(x) + iS(x)$.
- `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.

11.12.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.12.2 Macro Definition Documentation

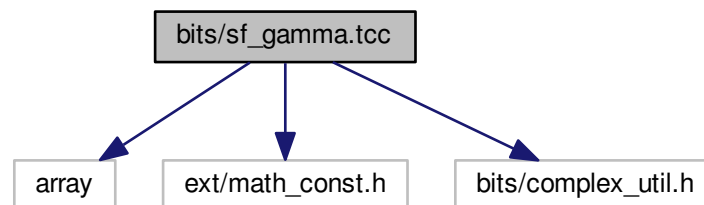
11.12.2.1 `#define _GLIBCXX_BITS_SF_FRESNEL_TCC 1`

Definition at line 31 of file `sf_fresnel.tcc`.

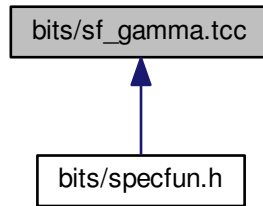
11.13 bits/sf_gamma.tcc File Reference

```
#include <array>
#include <ext/math_const.h>
#include <bits/complex_util.h>
```

Include dependency graph for `sf_gamma.tcc`:



This graph shows which files directly or indirectly include this file:



Classes

- struct `std::__detail::__gamma_lanczos_data< _Tp >`
- struct `std::__detail::__gamma_lanczos_data< double >`
- struct `std::__detail::__gamma_lanczos_data< float >`
- struct `std::__detail::__gamma_lanczos_data< long double >`
- struct `std::__detail::__gamma_spouge_data< _Tp >`
- struct `std::__detail::__gamma_spouge_data< double >`
- struct `std::__detail::__gamma_spouge_data< float >`
- struct `std::__detail::__gamma_spouge_data< long double >`
- struct `std::__detail::__Factorial_table< _Tp >`

Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_BITS_SF_GAMMA_TCC 1`

Functions

- template<typename `_Tp` >
`_Tp std::__detail::__binomial` (unsigned int `__n`, unsigned int `__k`)

Return the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

- `template<typename _Tp >`
`_Tp std::__detail::__binomial (_Tp __nu, unsigned int __k)`

Return the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu + 1)}{\Gamma(\nu - k + 1)\Gamma(k + 1)}$$

The binomial coefficients are generated by:

$$(1 + t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__double_factorial (int __n)`

Return the double factorial of the integer n .

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (unsigned int __n)`

Return the factorial of the integer n .

- `template<typename _Tp >`
`_Tp std::__detail::__falling_factorial (_Tp __a, int __n)`

Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n . The falling factorial function is defined by

$$a^{\underline{n}} = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $n^{\underline{n}} = n!$.

- `template<typename _Tp >`
`_Tp std::__detail::__falling_factorial (_Tp __a, _Tp __nu)`

Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and order ν . The falling factorial function is defined by

$$a^{\underline{\nu}} = \Gamma(a + 1) / \Gamma(a - \nu + 1)$$

- `template<typename _Tp >`
`_Tp std::__detail::__gamma (_Tp __a)`

Return the gamma function $\Gamma(a)$. The gamma function is defined by:

$$\Gamma(a) = \int_0^{\infty} e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`
`std::pair<_Tp, _Tp> std::__detail::__gamma (_Tp __a, _Tp __x)`

Return the incomplete gamma functions.

- `template<typename _Tp >`
`std::pair<_Tp, _Tp> std::__detail::__gamma_cont_frac (_Tp __a, _Tp __x)`

Return the incomplete gamma function by continued fraction.

- `template<typename _Tp >`
`_Tp std::__detail::__gamma_reciprocal (_Tp __a)`
- `template<typename _Tp >`
`_Tp std::__detail::__gamma_reciprocal_series (_Tp __a)`

- `template<typename _Tp >`
`std::pair<_Tp, _Tp> std::__detail::__gamma_series (_Tp __a, _Tp __x)`

Return the incomplete gamma function by series summation.

$$\gamma(a, x) = x^a e^{-x} \sum_{k=1}^{\infty} \frac{x^k}{(a)_k}$$

- template<typename _Tp >
_Tp std::__detail::__harmonic_number (unsigned int __n)
- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_binet1p (_Tp __z)

Return the Binet function $J(1 + z)$ by the Lanczos method. The Binet function is the log of the scaled Gamma function $\log(\Gamma^*(z))$ defined by

$$J(z) = \log(\Gamma^*(z)) = \log(\Gamma(z)) + z - \left(z - \frac{1}{2}\right) \log(z) - \log(2\pi)$$

or

$$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

where $\Gamma(z)$ is the gamma function.

- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_log_gamma1p (_Tp __z)

Return the logarithm of the gamma function $\log(\Gamma(1 + z))$ by the Lanczos method.

- template<typename _Tp >
_Tp std::__detail::__log_binomial (unsigned int __n, unsigned int __k)

Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

- template<typename _Tp >
_Tp std::__detail::__log_binomial (_Tp __nu, unsigned int __k)

Return the logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

- template<typename _Tp >
_Tp std::__detail::__log_binomial_sign (_Tp __nu, unsigned int __k)

Return the sign of the exponentiated logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu+1)}{\Gamma(\nu-k+1)\Gamma(k+1)}$$

The binomial coefficients are generated by:

$$(1+t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

- template<typename _Tp >
std::complex<_Tp> std::__detail::__log_binomial_sign (std::complex<_Tp> __nu, unsigned int __k)

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (_Tp __x)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (int __n)`

Return the logarithm of the double factorial of the integer n .

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_factorial (unsigned int __n)`

Return the logarithm of the factorial of the integer n .

- `template<typename _Tp >`
`_Tp std::__detail::__log_falling_factorial (_Tp __a, _Tp __nu)`

Return the logarithm of the falling factorial function or the lower Pochhammer symbol. The lower Pochhammer symbol is defined by

$$a^{\overline{n}} = \Gamma(a+1)/\Gamma(a-\nu+1) = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1$$

In particular, $n^{\overline{\{n\}}} = n!$. Thus this function returns

$$\ln[a^{\overline{n}}] = \ln[\Gamma(a+1)] - \ln[\Gamma(a-\nu+1)], \ln[a^{\overline{0}}] = 0$$

Many notations exist for this function:

$$(a)_{\nu}$$

,

$$\left\{ \begin{matrix} a \\ \nu \end{matrix} \right\}$$

, and others.

- `template<typename _Tp >`
`_Tp std::__detail::__log_gamma (_Tp __a)`

Return $\log(|\Gamma(a)|)$. This will return values even for $a < 0$. To recover the sign of $\Gamma(a)$ for any argument use `__log_gamma_sign`.

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__log_gamma (std::complex< _Tp > __a)`

Return $\log(\Gamma(a))$ for complex argument.

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (_Tp __x)`

Return $\log(\Gamma(x))$ by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.

- `template<typename _Tp >`
`_Tp std::__detail::__log_gamma_sign (_Tp __a)`

Return the sign of $\Gamma(x)$. At nonpositive integers zero is returned indicating $\Gamma(x)$ is undefined.

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__log_gamma_sign (std::complex< _Tp > __a)`

- `template<typename _Tp >`
`_Tp std::__detail::__log_rising_factorial (_Tp __a, _Tp __nu)`

Return the logarithm of the rising factorial function or the (upper) Pochhammer symbol. The Pochhammer symbol is defined for integer order by

$$a^{\overline{\nu}} = \Gamma(a+\nu)/\Gamma(a) = \prod_{k=0}^{\nu-1} (a+k), (a)_0 = 1$$

Thus this function returns

$$\ln[a^{\overline{\nu}}] = \ln[\Gamma(a+\nu)] - \ln[\Gamma(a)], \ln[(a)_0] = 0$$

Many notations exist for this function:

$$(a)_{\nu}$$

(especially in the literature of special functions),

$$\left[\begin{matrix} a \\ \nu \end{matrix} \right]$$

, 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::__psi (unsigned int __n)`

Return the digamma function of integral argument. The digamma or $\psi(x)$ function is defined as the logarithmic derivative of the gamma function:

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

The digamma series for integral argument is given by:

$$\psi(n) = -\gamma_E + \sum_{k=1}^{n-1} \frac{1}{k}$$

The latter sum is called the harmonic number, H_n .

- template<typename _Tp >

`_Tp std::__detail::__psi (_Tp __x)`

Return the digamma function. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

For negative argument the reflection formula is used:

$$\psi(x) = \psi(1-x) - \pi \cot(\pi x)$$

- template<typename _Tp >

`_Tp std::__detail::__psi (unsigned int __n, _Tp __x)`

Return the polygamma function $\psi^{(n)}(x)$.

- template<typename _Tp >

`_Tp std::__detail::__psi_asymp (_Tp __x)`

Return the digamma function for large argument. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

- template<typename _Tp >

`_Tp std::__detail::__psi_series (_Tp __x)`

Return the digamma function by series expansion. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

- template<typename _Tp >

`_Tp std::__detail::__qgamma (_Tp __a, _Tp __x)`

Return the regularized upper incomplete gamma function. The regularized upper incomplete gamma function is defined by

$$Q(a, x) = \frac{\Gamma(a, x)}{\Gamma(a)}$$

where $\Gamma(a)$ is the gamma function and

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1} dt (a > 0)$$

is the upper incomplete gamma function.

- `template<typename _Tp >`

`_Tp std::__detail::__rising_factorial (_Tp __a, int __n)`

Return the (upper) Pochhammer function or the rising factorial function. The Pochhammer symbol is defined by

$$a^{\overline{n}} = \Gamma(a + \nu) / \Gamma(\nu) = \prod_{k=0}^{n-1} (a + k), (a)_0 = 1$$

Many notations exist for this function:

$$(a)_\nu$$

, (especially in the literature of special functions),

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

- `template<typename _Tp >`

`_Tp std::__detail::__rising_factorial (_Tp __a, _Tp __nu)`

Return the rising factorial function or the (upper) Pochhammer function. The rising factorial function is defined by

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(\nu)$$

Many notations exist for this function:

$$(a)_\nu$$

, (especially in the literature of special functions),

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

- `template<typename _Tp >`

`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__spouge_binet1p (_Tp __z)`

Return the Binet function $J(1 + z)$ by the Spouge method. The Binet function is the log of the scaled Gamma function $\log(\Gamma^*(z))$ defined by

$$J(z) = \log(\Gamma^*(z)) = \log(\Gamma(z)) + z - \left(z - \frac{1}{2}\right) \log(z) - \log(2\pi)$$

or

$$\Gamma(z) = \sqrt{2\pi} z^{z-\frac{1}{2}} e^{-z} e^{J(z)}$$

where $\Gamma(z)$ is the gamma function.

- `template<typename _Tp >`

`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__spouge_log_gamma1p (_Tp __z)`

Return the logarithm of the gamma function $\log(\Gamma(1 + z))$ by the Spouge algorithm:

$$\Gamma(z + 1) = (z + a)^{z+1/2} e^{-z-a} \left[\sqrt{2\pi} + \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z + k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a - k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

.

- `template<typename _Tp >`
`_Tp std::__detail::__tgamma (_Tp __a, _Tp __x)`
Return the upper incomplete gamma function. The lower incomplete gamma function is defined by

$$\Gamma(a, x) = \int_x^{\infty} e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`
`_Tp std::__detail::__tgamma_lower (_Tp __a, _Tp __x)`
Return the lower incomplete gamma function. The lower incomplete gamma function is defined by

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

Variables

- `constexpr _Factorial_table< long double > std::__detail::__S_double_factorial_table [301]`
- `constexpr _Factorial_table< long double > std::__detail::__S_factorial_table [171]`
- `constexpr unsigned long long std::__detail::__S_harmonic_denom [_S_num_harmonic_number]`
- `constexpr unsigned long long std::__detail::__S_harmonic_number [_S_num_harmonic_number]`
- `constexpr _Factorial_table< long double > std::__detail::__S_neg_double_factorial_table [999]`
- `template<typename _Tp >`
`constexpr std::size_t std::__detail::__S_num_double_factorials = 0`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_double_factorials< double > = 301`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_double_factorials< float > = 57`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_double_factorials< long double > = 301`
- `template<typename _Tp >`
`constexpr std::size_t std::__detail::__S_num_factorials = 0`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_factorials< double > = 171`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_factorials< float > = 35`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_factorials< long double > = 171`
- `constexpr unsigned long long std::__detail::__S_num_harmonic_number = 29`
- `template<typename _Tp >`
`constexpr std::size_t std::__detail::__S_num_neg_double_factorials = 0`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_neg_double_factorials< double > = 150`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_neg_double_factorials< float > = 27`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_neg_double_factorials< long double > = 999`

11.13.1 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

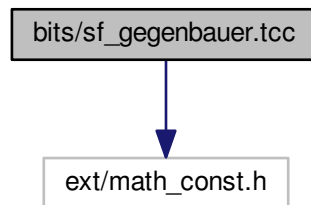
11.13.2.1 `#define _GLIBCXX_BITS_SF_GAMMA_TCC 1`

Definition at line 49 of file `sf_gamma.tcc`.

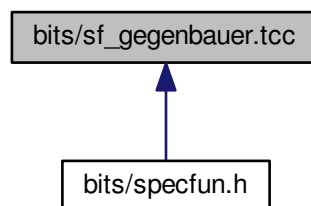
11.14 `bits/sf_gegenbauer.tcc` File Reference

```
#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)`

11.14.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

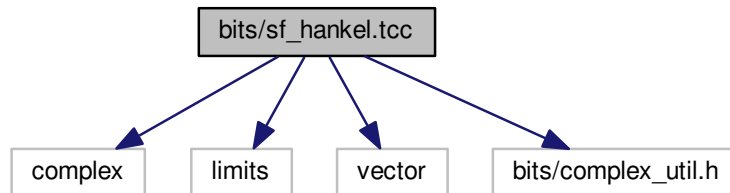
11.14.2 Macro Definition Documentation

11.14.2.1 `#define _GLIBCXX_BITS_SF_GEGENBAUER_TCC 1`

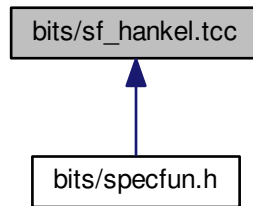
Definition at line 31 of file `sf_gegenbauer.tcc`.

11.15 bits/sf_hankel.tcc File Reference

```
#include <complex>
#include <limits>
#include <vector>
#include <bits/complex_util.h>
Include dependency graph for sf_hankel.tcc:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_HANKEL_TCC 1](#)

Functions

- `template<typename _Tp >`
`void std::__detail::__airy_arg (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)`
Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Bessel function.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_1 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Hankel function of the first kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_2 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_neumann (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Neumann function.
- `template<typename _Tp >`
`void std::__detail::__debye_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)`

- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn)`
- `template<typename _Tp >`
`void std::__detail::__hankel_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__nup2, std::complex< _Tp > &__num2, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__num4d3, std::complex< _Tp > &__zeta, std::complex< _Tp > &__zetaphf, std::complex< _Tp > &__zetamhf, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__zetrat)`
Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order nu along with their derivatives.
- `template<typename _Tp >`
`void std::__detail::__hankel_uniform_outer (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__etm3h, std::complex< _Tp > &__etrat, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)`
Compute outer factors and associated functions of z and nu appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of z and nu returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.
- `template<typename _Tp >`
`void std::__detail::__hankel_uniform_sum (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > __num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > __Aip, std::complex< _Tp > __o4dp, std::complex< _Tp > __Aim, std::complex< _Tp > __o4dm, std::complex< _Tp > __od2p, std::complex< _Tp > __od0dp, std::complex< _Tp > __od2m, std::complex< _Tp > __od0dm, _Tp __eps, std::complex< _Tp > &__H1sum, std::complex< _Tp > &__H1psum, std::complex< _Tp > &__H2sum, std::complex< _Tp > &__H2psum)`
Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to nterms (less than 5) to achieve relative error eps.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_bessel (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Bessel function.
- `template<typename _Tp >`
`__gnu_cxx::__sph_hankel_t< unsigned int, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__sph_hankel (unsigned int __n, std::complex< _Tp > __z)`
Helper to compute complex spherical Hankel functions and their derivatives.

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Hankel function of the first kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_neumann (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Neumann function.

11.15.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.15.2 Macro Definition Documentation

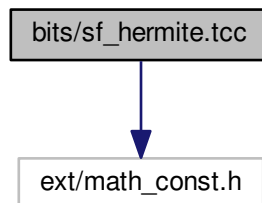
11.15.2.1 `#define _GLIBCXX_BITS_SF_HANKEL_TCC 1`

Definition at line 31 of file `sf_hankel.tcc`.

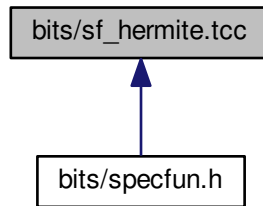
11.16 `bits/sf_hermite.tcc` File Reference

```
#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 > std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__hermite_zeros](#) (unsigned int __n, _Tp __proto=_Tp{})
- [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.
- [template<typename _Tp > _Tp std::__detail::__poly_prob_hermite_recursion](#) (unsigned int __n, _Tp __x)
This routine returns the Probabilists Hermite polynomial of order n: $He_n(x)$ by recursion on n.

11.16.1 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

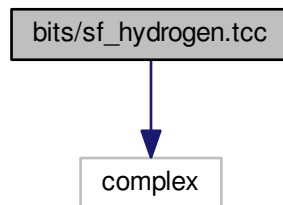
11.16.2.1 `#define _GLIBCXX_BITS_SF_HERMITE_TCC 1`

Definition at line 42 of file `sf_hermite.tcc`.

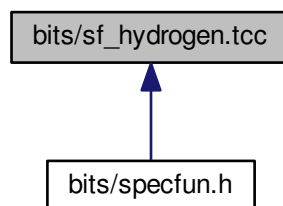
11.17 `bits/sf_hydrogen.tcc` File Reference

```
#include <complex>
```

Include dependency graph for `sf_hydrogen.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- `#define _GLIBCXX_BITS_SF_HYDROGEN_TCC 1`

Functions

- `template<typename _Tp >
std::complex< _Tp > std::__detail::__hydrogen (unsigned int __n, unsigned int __l, unsigned int __m, _Tp __Z,
_Tp __r, _Tp __theta, _Tp __phi)`

11.17.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.17.2 Macro Definition Documentation

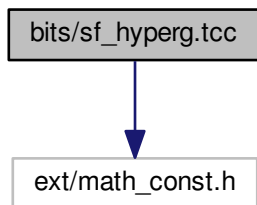
11.17.2.1 `#define _GLIBCXX_BITS_SF_HYDROGEN_TCC 1`

Definition at line 31 of file sf_hydrogen.tcc.

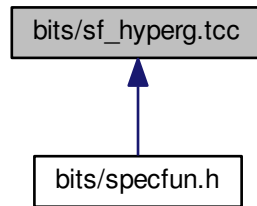
11.18 bits/sf_hyperg.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_hyperg.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_HYPERG_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__conf_hyperg \(_Tp __a, _Tp __c, _Tp __x\)](#)
Return the confluent hypergeometric function ${}_1F_1(a; c; x) = M(a, c, x)$.
- [template<typename _Tp > _Tp std::__detail::__conf_hyperg_lim \(_Tp __c, _Tp __x\)](#)
Return the confluent hypergeometric limit function ${}_0F_1(-; c; x)$.
- [template<typename _Tp > _Tp std::__detail::__conf_hyperg_lim_series \(_Tp __c, _Tp __x\)](#)
This routine returns the confluent hypergeometric limit function by series expansion.
- [template<typename _Tp > _Tp std::__detail::__conf_hyperg_luke \(_Tp __a, _Tp __c, _Tp __xin\)](#)
Return the hypergeometric function ${}_1F_1(a; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.
- [template<typename _Tp > _Tp std::__detail::__conf_hyperg_series \(_Tp __a, _Tp __c, _Tp __x\)](#)
This routine returns the confluent hypergeometric function by series expansion.
- [template<typename _Tp > _Tp std::__detail::__hyperg \(_Tp __a, _Tp __b, _Tp __c, _Tp __x\)](#)
Return the hypergeometric function ${}_2F_1(a, b; c; x)$.

- `template<typename _Tp >`
`_Tp std::__detail::__hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __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 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.
- `template<typename _Tp >`
`_Tp std::__detail::__tricoli_u (_Tp __a, _Tp __c, _Tp __x)`
Return the Tricoli confluent hypergeometric function

$$U(a, c, x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a; c; x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1; 2-c; x)$$

- `template<typename _Tp >`
`_Tp std::__detail::__tricoli_u_naive (_Tp __a, _Tp __c, _Tp __x)`
Return the Tricoli confluent hypergeometric function

$$U(a, c, x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a; c; x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1; 2-c; x)$$

11.18.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.18.2 Macro Definition Documentation

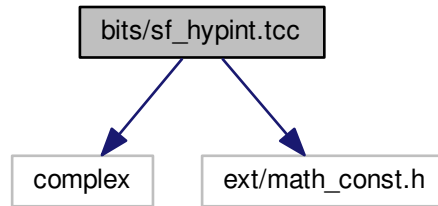
11.18.2.1 `#define GLIBCXX_BITS_SF_HYPERG_TCC 1`

Definition at line 44 of file `sf_hyperg.tcc`.

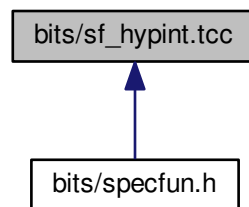
11.19 bits/sf_hypint.tcc File Reference

```
#include <complex>
#include <ext/math_const.h>
```

Include dependency graph for `sf_hypint.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_BITS_SF_HYPINT_TCC 1`

Functions

- `template<typename _Tp>`
`std::pair<_Tp, _Tp> std::__detail::__chshint(_Tp __x, _Tp &_Chi, _Tp &_Shi)`
This function returns the hyperbolic cosine $Ci(x)$ and hyperbolic sine $Si(x)$ integrals as a pair.

- `template<typename _Tp >`
`void std::__detail::__chshint_cont_frac (_Tp __t, _Tp &_Chi, _Tp &_Shi)`
This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by continued fraction for positive argument.
- `template<typename _Tp >`
`void std::__detail::__chshint_series (_Tp __t, _Tp &_Chi, _Tp &_Shi)`
This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by series summation for positive argument.

11.19.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.19.2 Macro Definition Documentation

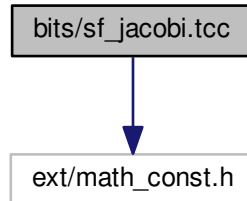
11.19.2.1 `#define _GLIBCXX_BITS_SF_HYPINT_TCC 1`

Definition at line 31 of file `sf_hypint.tcc`.

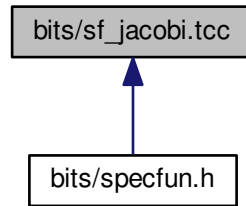
11.20 bits/sf_jacobi.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_jacobi.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_JACOBI_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__poly_jacobi \(unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__poly_radial_jacobi \(unsigned int __n, unsigned int __m, _Tp __rho\)](#)
- [template<typename _Tp > __gnu_cxx::__promote_fp_t< _Tp > std::__detail::__zernike \(unsigned int __n, int __m, _Tp __rho, _Tp __phi\)](#)

11.20.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

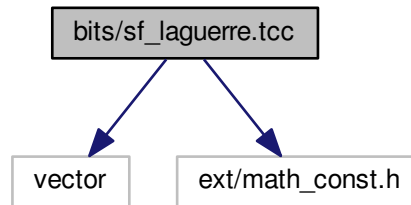
11.20.2 Macro Definition Documentation

11.20.2.1 [#define _GLIBCXX_BITS_SF_JACOBI_TCC 1](#)

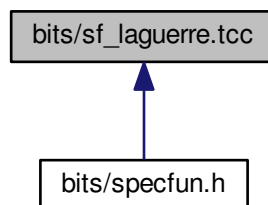
Definition at line 31 of file `sf_jacobi.tcc`.

11.21 bits/sf_laguerre.tcc File Reference

```
#include <vector>
#include <ext/math_const.h>
Include dependency graph for sf_laguerre.tcc:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__assoc_laguerre` (unsigned int __n, unsigned int __m, _Tp __x)
This routine returns the associated Laguerre polynomial of order n, degree m: $L_n^m(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__laguerre` (unsigned int __n, _Tp __x)
This routine returns the Laguerre polynomial of order n: $L_n(x)$.
- `template<typename _Tp >`
`std::vector<__gnu_cxx::__quadrature_point_t<_Tp>> std::__detail::__laguerre_zeros` (unsigned int __n, _Tp __alpha)
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` (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.

11.21.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.21.2 Macro Definition Documentation

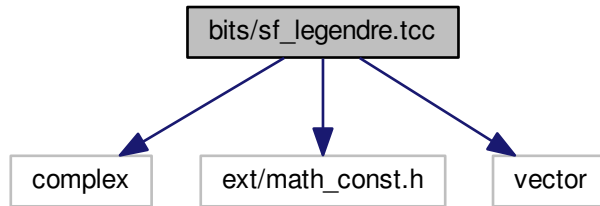
11.21.2.1 `#define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1`

Definition at line 44 of file `sf_laguerre.tcc`.

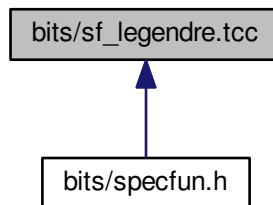
11.22 bits/sf_legendre.tcc File Reference

```
#include <complex>
#include <ext/math_const.h>
#include <vector>
```

Include dependency graph for sf_legendre.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__assoc_legendre_p \(unsigned int __l, unsigned int __m, _Tp __x\)](#)
Return the associated Legendre function by recursion on l and downward recursion on m .

- `template<typename _Tp >`
`_Tp std::__detail::__legendre_q (unsigned int __l, _Tp __x)`
Return the Legendre function of the second kind by upward recursion on order l.
- `template<typename _Tp >`
`std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__legendre_zeros (unsigned int __l, _Tp`
`proto=_Tp{})`
- `template<typename _Tp >`
`_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.

11.22.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

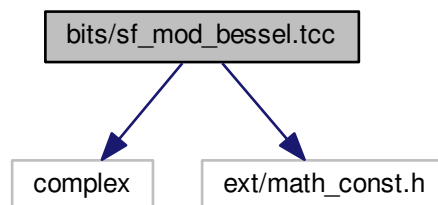
11.22.2 Macro Definition Documentation

11.22.2.1 `#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1`

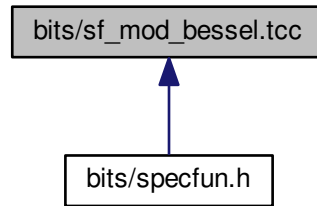
Definition at line 47 of file `sf_legendre.tcc`.

11.23 bits/sf_mod_bessel.tcc File Reference

```
#include <complex>
#include <ext/math_const.h>
Include dependency graph for sf_mod_bessel.tcc:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1](#)

Functions

- [template<typename _Tp > __gnu_cxx::__airy_t< _Tp, _Tp > std::__detail::__airy \(_Tp __z\)](#)
Compute the Airy functions $Ai(x)$ and $Bi(x)$ and their first derivatives $Ai'(x)$ and $Bi'(x)$ respectively.
- [template<typename _Tp > _Tp std::__detail::__cyl_bessel_i \(_Tp __nu, _Tp __x\)](#)
Return the regular modified Bessel function of order ν : $I_\nu(x)$.
- [template<typename _Tp > __gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > std::__detail::__cyl_bessel_ik \(_Tp __nu, _Tp __x\)](#)
Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.
- [template<typename _Tp > __gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > std::__detail::__cyl_bessel_ik_asymp \(_Tp __nu, _Tp __x\)](#)
This routine computes the asymptotic modified cylindrical Bessel and functions of order ν : $I_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.
- [template<typename _Tp > __gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > std::__detail::__cyl_bessel_ik_steel \(_Tp __nu, _Tp __x\)](#)
Compute the modified Bessel functions $I_\nu(x)$ and $K_\nu(x)$ and their first derivatives $I'_\nu(x)$ and $K'_\nu(x)$ respectively. These four functions are computed together for numerical stability.
- [template<typename _Tp > _Tp std::__detail::__cyl_bessel_k \(_Tp __nu, _Tp __x\)](#)
Return the irregular modified Bessel function $K_\nu(x)$ of order ν .

- `template<typename _Tp >`
`__gnu_cxx::__fock_airy_t< _Tp, std::complex< _Tp > > std::__detail::__fock_airy (_Tp __x)`
Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w_1'(x)$ and $w_2'(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$
- `template<typename _Tp >`
`__gnu_cxx::__sph_mod_bessel_t< unsigned int, _Tp, _Tp > std::__detail::__sph_bessel_ik (unsigned int __n, _Tp __x)`
Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i_n'(x)$ and $k_n'(x)$ respectively.

11.23.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

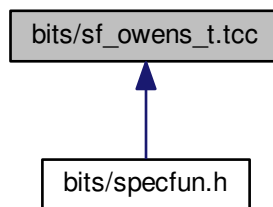
11.23.2 Macro Definition Documentation

11.23.2.1 `#define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1`

Definition at line 47 of file `sf_mod_bessel.tcc`.

11.24 bits/sf_owens_t.tcc File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_BITS_SF_OWENS_T_TCC 1`

Functions

- `template<typename _Tp >
_Tp std::__detail::__gauss (_Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__owens_t (_Tp __h, _Tp __a)`
- `template<typename _Tp >
_Tp std::__detail::__znorm1 (_Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__znorm2 (_Tp __x)`

11.24.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.24.2 Macro Definition Documentation

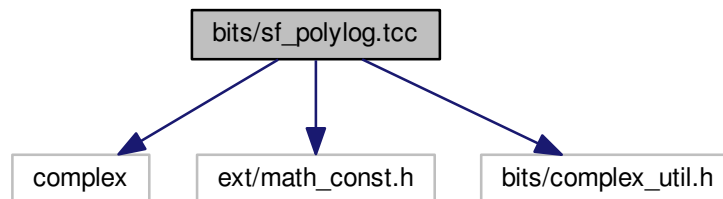
11.24.2.1 `#define _GLIBCXX_BITS_SF_OWENS_T_TCC 1`

Definition at line 31 of file `sf_owens_t.tcc`.

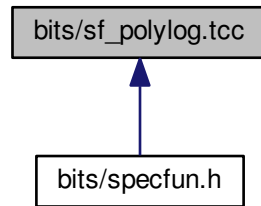
11.25 bits/sf_polylog.tcc File Reference

```
#include <complex>
#include <ext/math_const.h>
#include <bits/complex_util.h>
```

Include dependency graph for `sf_polylog.tcc`:



This graph shows which files directly or indirectly include this file:



Classes

- class [std::__detail::_AsympTerminator<_Tp>](#)
- class [std::__detail::_Terminator<_Tp>](#)

Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1](#)

Functions

- [template<typename _Sp, typename _Tp> _Tp std::__detail::__bose_einstein\(_Sp __s, _Tp __x\)](#)
- [template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_0_m2pi\(std::complex<_Tp> __z\)](#)
- [template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_pi\(std::complex<_Tp> __z\)](#)
- [template<typename _Tp> std::complex<_Tp> std::__detail::__clausen\(unsigned int __m, std::complex<_Tp> __z\)](#)
- [template<typename _Tp> _Tp std::__detail::__clausen\(unsigned int __m, _Tp __x\)](#)
- [template<typename _Tp> _Tp std::__detail::__clausen_cl\(unsigned int __m, std::complex<_Tp> __z\)](#)
- [template<typename _Tp> _Tp std::__detail::__clausen_cl\(unsigned int __m, _Tp __x\)](#)

- `template<typename _Tp >`
`_Tp std::__detail::__clausen_sl (unsigned int __m, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp std::__detail::__clausen_sl (unsigned int __m, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__dirichlet_beta (std::complex< _Tp > __s)`
- `template<typename _Tp >`
`_Tp std::__detail::__dirichlet_beta (_Tp __s)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__dirichlet_eta (std::complex< _Tp > __s)`
- `template<typename _Tp >`
`_Tp std::__detail::__dirichlet_eta (_Tp __s)`
- `template<typename _Tp >`
`_Tp std::__detail::__dirichlet_lambda (_Tp __s)`
- `template<typename _Sp, typename _Tp >`
`_Tp std::__detail::__fermi_dirac (_Sp __s, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__hurwitz_zeta_polylog (_Tp __s, std::complex< _Tp > __a)`
- `template<typename _Tp >`
`_Tp std::__detail::__polylog (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp, typename _ArgType >`
`__gnu_cxx::__promote_fp_t< std::complex< _Tp >, _ArgType > std::__detail::__polylog_exp (_Tp __s, _ArgType __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg (int __n, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_int (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_int (int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_real (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_real (_Tp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos_int (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos_int (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos_real (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos_real (_Tp __s, _Tp __w)`

- `template<typename _PowTp, typename _Tp >`
`_Tp std::__detail::__polylog_exp_sum (_PowTp __s, _Tp __w)`

11.25.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.25.2 Macro Definition Documentation

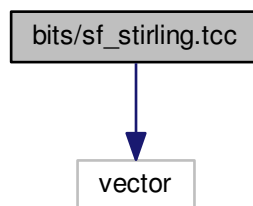
11.25.2.1 `#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1`

Definition at line 41 of file `sf_polylog.tcc`.

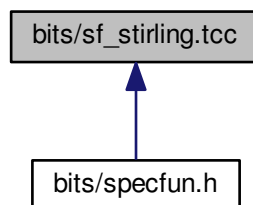
11.26 `bits/sf_stirling.tcc` File Reference

```
#include <vector>
```

Include dependency graph for `sf_stirling.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_STIRLING_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__log_stirling_1 \(unsigned int __n, unsigned int __m\)](#)
- [template<typename _Tp > _Tp std::__detail::__log_stirling_1_sign \(unsigned int __n, unsigned int __m\)](#)
- [template<typename _Tp > _Tp std::__detail::__log_stirling_2 \(unsigned int __n, unsigned int __m\)](#)
- [template<typename _Tp > _Tp std::__detail::__stirling_1 \(unsigned int __n, unsigned int __m\)](#)
- [template<typename _Tp > _Tp std::__detail::__stirling_1_recur \(unsigned int __n, unsigned int __m\)](#)
- [template<typename _Tp > _Tp std::__detail::__stirling_1_series \(unsigned int __n, unsigned int __m\)](#)
- [template<typename _Tp > _Tp std::__detail::__stirling_2 \(unsigned int __n, unsigned int __m\)](#)
- [template<typename _Tp > _Tp std::__detail::__stirling_2_recur \(unsigned int __n, unsigned int __m\)](#)
- [template<typename _Tp > _Tp std::__detail::__stirling_2_series \(unsigned int __n, unsigned int __m\)](#)

11.26.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.26.2 Macro Definition Documentation

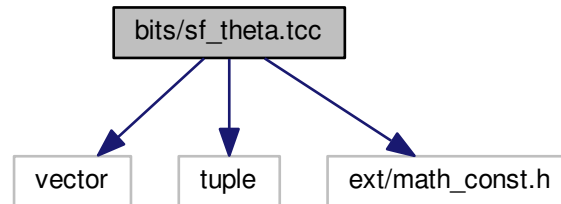
11.26.2.1 [#define _GLIBCXX_BITS_SF_STIRLING_TCC 1](#)

Definition at line 35 of file `sf_stirling.tcc`.

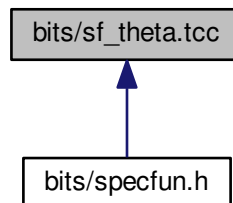
11.27 bits/sf_theta.tcc File Reference

```
#include <vector>
#include <tuple>
#include <ext/math_const.h>
```

Include dependency graph for sf_theta.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_THETA_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__ellnome (_Tp __k)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellnome_k (_Tp __k)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellnome_series (_Tp __k)`
- `template<typename _Tp >`
`__gnu_cxx::__jacobi_t< _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)`

11.27.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.27.2 Macro Definition Documentation

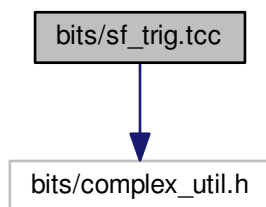
11.27.2.1 `#define _GLIBCXX_BITS_SF_THETA_TCC 1`

Definition at line 31 of file `sf_theta.tcc`.

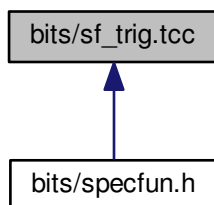
11.28 bits/sf_trig.tcc File Reference

```
#include <bits/complex_util.h>
```

Include dependency graph for sf_trig.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_TRIG_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__cos_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cos_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp std::__detail::__cosh_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cosh_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polar_pi (_Tp __rho, _Tp __phi_pi)`
- `template<typename _Tp >`
`_Tp std::__detail::__sin_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sin_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< _Tp > std::__detail::__sincos (_Tp __x)`
- `template<>`
`__gnu_cxx::__sincos_t< float > std::__detail::__sincos (float __x)`
- `template<>`
`__gnu_cxx::__sincos_t< double > std::__detail::__sincos (double __x)`
- `template<>`
`__gnu_cxx::__sincos_t< long double > std::__detail::__sincos (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< _Tp > std::__detail::__sincos_pi (_Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__sinh_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sinh_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp std::__detail::__tan_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__tan_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp std::__detail::__tanh_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__tanh_pi (std::complex< _Tp > __z)`

11.28.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

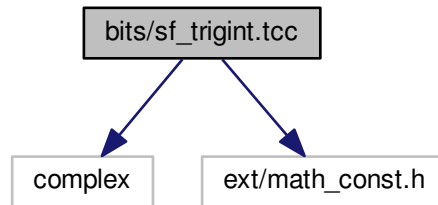
11.28.2 Macro Definition Documentation

11.28.2.1 `#define _GLIBCXX_BITS_SF_TRIG_TCC 1`

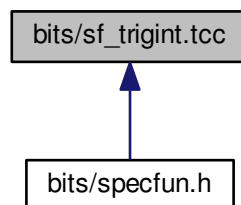
Definition at line 31 of file sf_trig.tcc.

11.29 bits/sf_trigint.tcc File Reference

```
#include <complex>
#include <ext/math_const.h>
Include dependency graph for sf_trigint.tcc:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_BITS_SF_TRIGINT_TCC 1`

Functions

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > std::__detail::__sincosint (_Tp __x)`
This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a pair.
- `template<typename _Tp >`
`void std::__detail::__sincosint_asymp (_Tp __t, _Tp &_Si, _Tp &_Ci)`
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.
- `template<typename _Tp >`
`void std::__detail::__sincosint_cont_frac (_Tp __t, _Tp &_Si, _Tp &_Ci)`
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.
- `template<typename _Tp >`
`void std::__detail::__sincosint_series (_Tp __t, _Tp &_Si, _Tp &_Ci)`
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.

11.29.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.29.2 Macro Definition Documentation

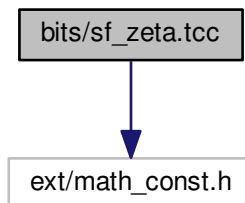
11.29.2.1 `#define _GLIBCXX_BITS_SF_TRIGINT_TCC 1`

Definition at line 31 of file `sf_trigint.tcc`.

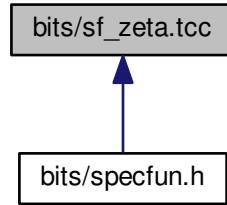
11.30 bits/sf_zeta.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for `sf_zeta.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_ZETA_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__debye \(unsigned int __n, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__dilog \(_Tp __x\)](#)
Compute the dilogarithm function $Li_2(x)$ by summation for $x \leq 1$.
- [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_euler_maclaurin \(_Tp __s\)](#)
Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.
- [template<typename _Tp > _Tp std::__detail::__riemann_zeta_glob \(_Tp __s\)](#)

- `template<typename _Tp >`
`_Tp std::__detail::__riemann_zeta_m_1 (_Tp __s)`
Return the Riemann zeta function $\zeta(s) - 1$.
- `template<typename _Tp >`
`_Tp std::__detail::__riemann_zeta_m_1_glob (_Tp __s)`
Evaluate the Riemann zeta function by series for all $s \neq 1$. Convergence is great until largish negative numbers. Then the convergence of the > 0 sum gets better.
- `template<typename _Tp >`
`_Tp std::__detail::__riemann_zeta_product (_Tp __s)`
Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.
- `template<typename _Tp >`
`_Tp std::__detail::__riemann_zeta_sum (_Tp __s)`
Compute the Riemann zeta function $\zeta(s)$ by summation for $s > 1$.

Variables

- `constexpr size_t std::__detail::_Num_Euler_Maclaurin_zeta = 100`
- `constexpr long double std::__detail::_S_Euler_Maclaurin_zeta [_Num_Euler_Maclaurin_zeta]`
- `constexpr size_t std::__detail::_S_num_zetam1 = 121`
- `constexpr long double std::__detail::_S_zetam1 [_S_num_zetam1]`

11.30.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.30.2 Macro Definition Documentation

11.30.2.1 `#define _GLIBCXX_BITS_SF_ZETA_TCC 1`

Definition at line 46 of file `sf_zeta.tcc`.

11.31 bits/specfun.h File Reference

```
#include <bits/c++config.h>
#include <limits>
#include <bits/stl_algobase.h>
#include <bits/specfun_state.h>
#include <bits/specfun_util.h>
#include <type_traits>
#include <bits/numeric_limits.h>
#include <bits/complex_util.h>
#include <bits/sf_trig.tcc>
#include <bits/sf_bernoulli.tcc>
#include <bits/sf_gamma.tcc>
#include <bits/sf_euler.tcc>
#include <bits/sf_stirling.tcc>
#include <bits/sf_bessel.tcc>
#include <bits/sf_beta.tcc>
#include <bits/sf_cardinal.tcc>
#include <bits/sf_chebyshev.tcc>
#include <bits/sf_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

Functions

- template<typename `_Tp` >
`__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::airy_ai(_Tp __x)`
- template<typename `_Tp` >
`std::complex<__gnu_cxx::__promote_fp_t<_Tp>> __gnu_cxx::airy_ai(std::complex<_Tp> __x)`
- float `__gnu_cxx::airy_aif(float __x)`
- long double `__gnu_cxx::airy_ail(long double __x)`
- template<typename `_Tp` >
`__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::airy_bi(_Tp __x)`
- template<typename `_Tp` >
`std::complex<__gnu_cxx::__promote_fp_t<_Tp>> __gnu_cxx::airy_bi(std::complex<_Tp> __x)`
- float `__gnu_cxx::airy_bif(float __x)`
- long double `__gnu_cxx::airy_bil(long double __x)`
- template<typename `_Tp` >
`__gnu_cxx::__promote_fp_t<_Tp> std::assoc_laguerre(unsigned int __n, unsigned int __m, _Tp __x)`
- float `std::assoc_laguerref(unsigned int __n, unsigned int __m, float __x)`
- long double `std::assoc_laguerrel(unsigned int __n, unsigned int __m, long double __x)`
- template<typename `_Tp` >
`__gnu_cxx::__promote_fp_t<_Tp> std::assoc_legendre(unsigned int __l, unsigned int __m, _Tp __x)`
- float `std::assoc_legendref(unsigned int __l, unsigned int __m, float __x)`
- long double `std::assoc_legendrel(unsigned int __l, unsigned int __m, long double __x)`
- template<typename `_Tp` >
`__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::bernoulli(unsigned int __n)`
- template<typename `_Tp` >
`_Tp __gnu_cxx::bernoulli(unsigned int __n, _Tp __x)`
- float `__gnu_cxx::bernoullif(unsigned int __n)`
- long double `__gnu_cxx::bernoullil(unsigned int __n)`
- template<typename `_Tpa`, typename `_Tpb` >
`__gnu_cxx::__promote_fp_t<_Tpa, _Tpb> std::beta(_Tpa __a, _Tpb __b)`
- float `std::betaf(float __a, float __b)`
- long double `std::betal(long double __a, long double __b)`
- template<typename `_Tp` >
`__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::binomial(unsigned int __n, unsigned int __k)`

Return the binomial coefficient as a real number. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

- template<typename `_Tp` >
`__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::binomial_cdf(_Tp __p, unsigned int __n, unsigned int __k)`

Return the binomial cumulative distribution function.

- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::binomial_pdf (_Tp __p, unsigned int __n, unsigned int __k)`
Return the binomial probability mass function.
- `float __gnu_cxx::binomialf (unsigned int __n, unsigned int __k)`
- `long double __gnu_cxx::binomiall (unsigned int __n, unsigned int __k)`
- `template<typename _Tps, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tps, _Tp > __gnu_cxx::bose_einstein (_Tps __s, _Tp __x)`
- `float __gnu_cxx::bose_einsteinf (float __s, float __x)`
- `long double __gnu_cxx::bose_einsteinl (long double __s, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::chebyshev_t (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::chebyshev_tf (unsigned int __n, float __x)`
- `long double __gnu_cxx::chebyshev_tl (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::chebyshev_u (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::chebyshev_uf (unsigned int __n, float __x)`
- `long double __gnu_cxx::chebyshev_ul (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::chebyshev_v (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::chebyshev_vf (unsigned int __n, float __x)`
- `long double __gnu_cxx::chebyshev_vl (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::chebyshev_w (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::chebyshev_wf (unsigned int __n, float __x)`
- `long double __gnu_cxx::chebyshev_wl (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::clausen (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::clausen (unsigned int __m, std::complex< __Tp > __z)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::clausen_cl (unsigned int __m, _Tp __x)`
- `float __gnu_cxx::clausen_clf (unsigned int __m, float __x)`
- `long double __gnu_cxx::clausen_cll (unsigned int __m, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::clausen_sl (unsigned int __m, _Tp __x)`
- `float __gnu_cxx::clausen_slf (unsigned int __m, float __x)`
- `long double __gnu_cxx::clausen_sll (unsigned int __m, long double __x)`
- `float __gnu_cxx::clausenf (unsigned int __m, float __x)`
- `std::complex< float > __gnu_cxx::clausenf (unsigned int __m, std::complex< float > __z)`
- `long double __gnu_cxx::clausenl (unsigned int __m, long double __x)`
- `std::complex< long double > __gnu_cxx::clausenl (unsigned int __m, std::complex< long double > __z)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::comp_ellint_1 (_Tp __k)`
- `float std::comp_ellint_1f (float __k)`
- `long double std::comp_ellint_1l (long double __k)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::comp_ellint_2 (_Tp __k)`
- `float std::comp_ellint_2f (float __k)`
- `long double std::comp_ellint_2l (long double __k)`
- `template<typename _Tp, typename _Tpn >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tpn > std::comp_ellint_3 (_Tp __k, _Tpn __nu)`

- float [std::comp_ellint_3f](#) (float __k, float __nu)
Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for float modulus k.
- long double [std::comp_ellint_3l](#) (long double __k, long double __nu)
Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for long double modulus k.
- template<typename _Tk >
[__gnu_cxx::__promote_fp_t<_Tk> __gnu_cxx::comp_ellint_d](#) (_Tk __k)
- float [__gnu_cxx::comp_ellint_df](#) (float __k)
- long double [__gnu_cxx::comp_ellint_dl](#) (long double __k)
- float [__gnu_cxx::comp_ellint_rf](#) (float __x, float __y)
- long double [__gnu_cxx::comp_ellint_rf](#) (long double __x, long double __y)
- template<typename _Tx, typename _Ty >
[__gnu_cxx::__promote_fp_t<_Tx, _Ty> __gnu_cxx::comp_ellint_rf](#) (_Tx __x, _Ty __y)
- float [__gnu_cxx::comp_ellint_rg](#) (float __x, float __y)
- long double [__gnu_cxx::comp_ellint_rg](#) (long double __x, long double __y)
- template<typename _Tx, typename _Ty >
[__gnu_cxx::__promote_fp_t<_Tx, _Ty> __gnu_cxx::comp_ellint_rg](#) (_Tx __x, _Ty __y)
- template<typename _Tpa, typename _Tpc, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tpa, _Tpc, _Tp> __gnu_cxx::conf_hyperg](#) (_Tpa __a, _Tpc __c, _Tp __x)
- template<typename _Tpc, typename _Tp >
[__gnu_cxx::__promote_2<_Tpc, _Tp>::__type __gnu_cxx::conf_hyperg_lim](#) (_Tpc __c, _Tp __x)
- float [__gnu_cxx::conf_hyperg_limf](#) (float __c, float __x)
- long double [__gnu_cxx::conf_hyperg_liml](#) (long double __c, long double __x)
- float [__gnu_cxx::conf_hypergf](#) (float __a, float __c, float __x)
- long double [__gnu_cxx::conf_hypergl](#) (long double __a, long double __c, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::cos_pi](#) (_Tp __x)
- float [__gnu_cxx::cos_pif](#) (float __x)
- long double [__gnu_cxx::cos_pil](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::cosh_pi](#) (_Tp __x)
- float [__gnu_cxx::cosh_pif](#) (float __x)
- long double [__gnu_cxx::cosh_pil](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::coshint](#) (_Tp __x)
- float [__gnu_cxx::coshintf](#) (float __x)
- long double [__gnu_cxx::coshintl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp> __gnu_cxx::cosint](#) (_Tp __x)
- float [__gnu_cxx::cosintf](#) (float __x)
- long double [__gnu_cxx::cosintl](#) (long double __x)
- template<typename _Tpnu, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> std::cyl_bessel_i](#) (_Tpnu __nu, _Tp __x)
- float [std::cyl_bessel_if](#) (float __nu, float __x)
- long double [std::cyl_bessel_il](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> std::cyl_bessel_j](#) (_Tpnu __nu, _Tp __x)
- float [std::cyl_bessel_jf](#) (float __nu, float __x)
- long double [std::cyl_bessel_jl](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tpnu, _Tp> std::cyl_bessel_k](#) (_Tpnu __nu, _Tp __x)
- float [std::cyl_bessel_kf](#) (float __nu, float __x)

- long double [std::cyl_bessel_kl](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp>
std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp> > [__gnu_cxx::cyl_hankel_1](#) (_Tpnu __nu, _Tp __z)
- template<typename _Tpnu, typename _Tp>
std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp> > [__gnu_cxx::cyl_hankel_1](#) (std::complex< _Tpnu > __nu, std::complex< _Tp> __x)
- std::complex< float > [__gnu_cxx::cyl_hankel_1f](#) (float __nu, float __z)
- std::complex< float > [__gnu_cxx::cyl_hankel_1f](#) (std::complex< float > __nu, std::complex< float > __x)
- std::complex< long double > [__gnu_cxx::cyl_hankel_1l](#) (long double __nu, long double __z)
- std::complex< long double > [__gnu_cxx::cyl_hankel_1l](#) (std::complex< long double > __nu, std::complex< long double > __x)
- template<typename _Tpnu, typename _Tp>
std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp> > [__gnu_cxx::cyl_hankel_2](#) (_Tpnu __nu, _Tp __z)
- template<typename _Tpnu, typename _Tp>
std::complex< __gnu_cxx::__promote_fp_t< _Tpnu, _Tp> > [__gnu_cxx::cyl_hankel_2](#) (std::complex< _Tpnu > __nu, std::complex< _Tp> __x)
- std::complex< float > [__gnu_cxx::cyl_hankel_2f](#) (float __nu, float __z)
- std::complex< float > [__gnu_cxx::cyl_hankel_2f](#) (std::complex< float > __nu, std::complex< float > __x)
- std::complex< long double > [__gnu_cxx::cyl_hankel_2l](#) (long double __nu, long double __z)
- std::complex< long double > [__gnu_cxx::cyl_hankel_2l](#) (std::complex< long double > __nu, std::complex< long double > __x)
- template<typename _Tpnu, typename _Tp>
__gnu_cxx::__promote_fp_t< _Tpnu, _Tp> [std::cyl_neumann](#) (_Tpnu __nu, _Tp __x)
- float [std::cyl_neumannf](#) (float __nu, float __x)
- long double [std::cyl_neumannl](#) (long double __nu, long double __x)
- template<typename _Tp>
__gnu_cxx::__promote_fp_t< _Tp> [__gnu_cxx::dawson](#) (_Tp __x)
- float [__gnu_cxx::dawsonf](#) (float __x)
- long double [__gnu_cxx::dawsonl](#) (long double __x)
- template<typename _Tp>
__gnu_cxx::__promote_fp_t< _Tp> [__gnu_cxx::debye](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::debyef](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::debyel](#) (unsigned int __n, long double __x)
- template<typename _Tp>
__gnu_cxx::__promote_fp_t< _Tp> [__gnu_cxx::dilog](#) (_Tp __x)
- float [__gnu_cxx::dilogf](#) (float __x)
- long double [__gnu_cxx::dilogl](#) (long double __x)
- template<typename _Tp>
_Tp [__gnu_cxx::dirichlet_beta](#) (_Tp __s)
- float [__gnu_cxx::dirichlet_betaf](#) (float __s)
- long double [__gnu_cxx::dirichlet_betel](#) (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_etel](#) (long double __s)
- template<typename _Tp>
_Tp [__gnu_cxx::dirichlet_lambda](#) (_Tp __s)
- float [__gnu_cxx::dirichlet_lambdaf](#) (float __s)
- long double [__gnu_cxx::dirichlet_lambdal](#) (long double __s)
- template<typename _Tp>
__gnu_cxx::__promote_fp_t< _Tp> [__gnu_cxx::double_factorial](#) (int __n)

Return the double factorial $n!!$ of the argument as a real number.

$$n!! = n(n-2)\dots(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)\dots(1), (-1)!! = 1$$

for odd n .

- float [__gnu_cxx::double_factorialf](#) (int __n)
- long double [__gnu_cxx::double_factoriall](#) (int __n)
- template<typename _Tp, typename _Tpp >
 [__gnu_cxx::__promote_fp_t](#)< _Tp, _Tpp > [std::ellint_1](#) (_Tp __k, _Tpp __phi)
- float [std::ellint_1f](#) (float __k, float __phi)
- long double [std::ellint_1l](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Tpp >
 [__gnu_cxx::__promote_fp_t](#)< _Tp, _Tpp > [std::ellint_2](#) (_Tp __k, _Tpp __phi)
- float [std::ellint_2f](#) (float __k, float __phi)

Return the incomplete elliptic integral of the second kind $E(k, \phi)$ for float argument.

- long double [std::ellint_2l](#) (long double __k, long double __phi)

Return the incomplete elliptic integral of the second kind $E(k, \phi)$.

- template<typename _Tp, typename _Tpn, typename _Tpp >
 [__gnu_cxx::__promote_fp_t](#)< _Tp, _Tpn, _Tpp > [std::ellint_3](#) (_Tp __k, _Tpn __nu, _Tpp __phi)

Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.

- float [std::ellint_3f](#) (float __k, float __nu, float __phi)

Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ for float argument.

- long double [std::ellint_3l](#) (long double __k, long double __nu, long double __phi)

Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.

- template<typename _Tk, typename _Tp, typename _Ta, typename _Tb >
 [__gnu_cxx::__promote_fp_t](#)< _Tk, _Tp, _Ta, _Tb > [__gnu_cxx::ellint_cel](#) (_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)
- float [__gnu_cxx::ellint_celf](#) (float __k_c, float __p, float __a, float __b)
- long double [__gnu_cxx::ellint_cell](#) (long double __k_c, long double __p, long double __a, long double __b)
- template<typename _Tk, typename _Tphi >
 [__gnu_cxx::__promote_fp_t](#)< _Tk, _Tphi > [__gnu_cxx::ellint_d](#) (_Tk __k, _Tphi __phi)
- float [__gnu_cxx::ellint_df](#) (float __k, float __phi)
- long double [__gnu_cxx::ellint_dl](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Tk >
 [__gnu_cxx::__promote_fp_t](#)< _Tp, _Tk > [__gnu_cxx::ellint_el1](#) (_Tp __x, _Tk __k_c)
- float [__gnu_cxx::ellint_el1f](#) (float __x, float __k_c)
- long double [__gnu_cxx::ellint_el1l](#) (long double __x, long double __k_c)
- template<typename _Tp, typename _Tk, typename _Ta, typename _Tb >
 [__gnu_cxx::__promote_fp_t](#)< _Tp, _Tk, _Ta, _Tb > [__gnu_cxx::ellint_el2](#) (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)
- float [__gnu_cxx::ellint_el2f](#) (float __x, float __k_c, float __a, float __b)
- long double [__gnu_cxx::ellint_el2l](#) (long double __x, long double __k_c, long double __a, long double __b)
- template<typename _Tx, typename _Tk, typename _Tp >
 [__gnu_cxx::__promote_fp_t](#)< _Tx, _Tk, _Tp > [__gnu_cxx::ellint_el3](#) (_Tx __x, _Tk __k_c, _Tp __p)
- float [__gnu_cxx::ellint_el3f](#) (float __x, float __k_c, float __p)
- long double [__gnu_cxx::ellint_el3l](#) (long double __x, long double __k_c, long double __p)
- template<typename _Tp, typename _Up >
 [__gnu_cxx::__promote_fp_t](#)< _Tp, _Up > [__gnu_cxx::ellint_rc](#) (_Tp __x, _Up __y)
- float [__gnu_cxx::ellint_rcf](#) (float __x, float __y)
- long double [__gnu_cxx::ellint_rcl](#) (long double __x, long double __y)

- `template<typename _Tp, typename _Up, typename _Vp >`
`__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rd (_Tp __x, _Up __y, _Vp __z)`
- `float __gnu_cxx::ellint_rdf (float __x, float __y, float __z)`
- `long double __gnu_cxx::ellint_rdl (long double __x, long double __y, long double __z)`
- `template<typename _Tp, typename _Up, typename _Vp >`
`__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rf (_Tp __x, _Up __y, _Vp __z)`
- `float __gnu_cxx::ellint_rff (float __x, float __y, float __z)`
- `long double __gnu_cxx::ellint_rfl (long double __x, long double __y, long double __z)`
- `template<typename _Tp, typename _Up, typename _Vp >`
`__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rg (_Tp __x, _Up __y, _Vp __z)`
- `float __gnu_cxx::ellint_rgf (float __x, float __y, float __z)`
- `long double __gnu_cxx::ellint_rgl (long double __x, long double __y, long double __z)`
- `template<typename _Tp, typename _Up, typename _Vp, typename _Wp >`
`__gnu_cxx::__promote_fp_t< _Tp, _Up, _Vp, _Wp > __gnu_cxx::ellint_rj (_Tp __x, _Up __y, _Vp __z, _Wp __p)`
- `float __gnu_cxx::ellint_rjf (float __x, float __y, float __z, float __p)`
- `long double __gnu_cxx::ellint_rjl (long double __x, long double __y, long double __z, long double __p)`
- `template<typename _Tp >`
`_Tp __gnu_cxx::ellnome (_Tp __k)`
- `float __gnu_cxx::ellnomef (float __k)`
- `long double __gnu_cxx::ellnomel (long double __k)`
- `template<typename _Tp >`
`_Tp __gnu_cxx::euler (unsigned int __n)`
This returns Euler number E_n .
- `template<typename _Tp >`
`_Tp __gnu_cxx::eulerian_1 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __gnu_cxx::eulerian_2 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::expint (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::expint (unsigned int __n, _Tp __x)`
- `float std::expintf (float __x)`
- `float __gnu_cxx::expintf (unsigned int __n, float __x)`
- `long double std::expintl (long double __x)`
- `long double __gnu_cxx::expintl (unsigned int __n, long double __x)`
- `template<typename _Tlam, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tlam, _Tp > __gnu_cxx::exponential_cdf (_Tlam __lambda, _Tp __x)`
Return the exponential cumulative probability density function.
- `template<typename _Tlam, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tlam, _Tp > __gnu_cxx::exponential_pdf (_Tlam __lambda, _Tp __x)`
Return the exponential probability density function.
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::factorial (unsigned int __n)`
Return the factorial $n!$ of the argument as a real number.

$$n! = 1 \times 2 \times \dots \times n, 0! = 1$$
- `float __gnu_cxx::factorialf (unsigned int __n)`
- `long double __gnu_cxx::factoriall (unsigned int __n)`
- `template<typename _Tp, typename _Tnu >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::falling_factorial (_Tp __a, _Tnu __nu)`

Return the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n . The falling factorial function is defined by

$$a^{\underline{n}} = \prod_{k=0}^{n-1} (a - k), a^{\underline{0}} = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $n^{\underline{n}} = n!$.

- float [__gnu_cxx::falling_factorialf](#) (float __a, float __nu)
- long double [__gnu_cxx::falling_factoriall](#) (long double __a, long double __nu)
- template<typename _Tps, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tps, _Tp>](#) [__gnu_cxx::fermi_dirac](#) (_Tps __s, _Tp __x)
- float [__gnu_cxx::fermi_diracf](#) (float __s, float __x)
- long double [__gnu_cxx::fermi_diracl](#) (long double __s, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::fisher_f_cdf](#) (_Tp __F, unsigned int __nu1, unsigned int __nu2)
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::fisher_f_pdf](#) (_Tp __F, unsigned int __nu1, unsigned int __nu2)
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::fresnel_c](#) (_Tp __x)
- float [__gnu_cxx::fresnel_csf](#) (float __x)
- long double [__gnu_cxx::fresnel_csl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::fresnel_s](#) (_Tp __x)
- float [__gnu_cxx::fresnel_ssf](#) (float __x)
- long double [__gnu_cxx::fresnel_ssl](#) (long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp>](#) [__gnu_cxx::gamma_cdf](#) (_Ta __alpha, _Tb __beta, _Tp __x)
Return the gamma cumulative propability distribution function.
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Ta, _Tb, _Tp>](#) [__gnu_cxx::gamma_pdf](#) (_Ta __alpha, _Tb __beta, _Tp __x)
Return the gamma propability distribution function.
- template<typename _Ta >
[__gnu_cxx::__promote_fp_t<_Ta>](#) [__gnu_cxx::gamma_reciprocal](#) (_Ta __a)
- float [__gnu_cxx::gamma_reciprocalf](#) (float __a)
- long double [__gnu_cxx::gamma_reciprocall](#) (long double __a)
- template<typename _Talpha, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Talpha, _Tp>](#) [__gnu_cxx::gegenbauer](#) (unsigned int __n, _Talpha __alpha, _Tp __x)
- float [__gnu_cxx::gegenbauerf](#) (unsigned int __n, float __alpha, float __x)
- long double [__gnu_cxx::gegenbauerl](#) (unsigned int __n, long double __alpha, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::harmonic](#) (unsigned int __n)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [std::hermite](#) (unsigned int __n, _Tp __x)
- float [std::hermitef](#) (unsigned int __n, float __x)
- long double [std::hermitel](#) (unsigned int __n, long double __x)
- template<typename _Tk, typename _Tphi >
[__gnu_cxx::__promote_fp_t<_Tk, _Tphi>](#) [__gnu_cxx::heuman_lambda](#) (_Tk __k, _Tphi __phi)

- float [__gnu_cxx::heuman_lambdaf](#) (float __k, float __phi)
- long double [__gnu_cxx::heuman_lambdal](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Up >
[__gnu_cxx::__promote_fp_t](#)< _Tp, _Up > [__gnu_cxx::hurwitz_zeta](#) (_Tp __s, _Up __a)
- template<typename _Tp, typename _Up >
std::complex< _Tp > [__gnu_cxx::hurwitz_zeta](#) (_Tp __s, std::complex< _Up > __a)
- float [__gnu_cxx::hurwitz_zetaf](#) (float __s, float __a)
- long double [__gnu_cxx::hurwitz_zetal](#) (long double __s, long double __a)
- template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tpa, _Tpb, _Tpc, _Tp > [__gnu_cxx::hyperg](#) (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)
- float [__gnu_cxx::hypergf](#) (float __a, float __b, float __c, float __x)
- long double [__gnu_cxx::hypergl](#) (long double __a, long double __b, long double __c, long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Ta, _Tb, _Tp > [__gnu_cxx::ibeta](#) (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Ta, _Tb, _Tp > [__gnu_cxx::ibetac](#) (_Ta __a, _Tb __b, _Tp __x)
- float [__gnu_cxx::ibetacf](#) (float __a, float __b, float __x)
- long double [__gnu_cxx::ibetacl](#) (long double __a, long double __b, long double __x)
- float [__gnu_cxx::ibetaf](#) (float __a, float __b, float __x)
- long double [__gnu_cxx::ibetal](#) (long double __a, long double __b, long double __x)
- template<typename _Talpha, typename _Tbeta, typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Talpha, _Tbeta, _Tp > [__gnu_cxx::jacobi](#) (unsigned __n, _Talpha __alpha, \leftrightarrow _Tbeta __beta, _Tp __x)
- template<typename _Kp, typename _Up >
[__gnu_cxx::__promote_fp_t](#)< _Kp, _Up > [__gnu_cxx::jacobi_cn](#) (_Kp __k, _Up __u)
- float [__gnu_cxx::jacobi_cnf](#) (float __k, float __u)
- long double [__gnu_cxx::jacobi_cnl](#) (long double __k, long double __u)
- template<typename _Kp, typename _Up >
[__gnu_cxx::__promote_fp_t](#)< _Kp, _Up > [__gnu_cxx::jacobi_dn](#) (_Kp __k, _Up __u)
- float [__gnu_cxx::jacobi_dnf](#) (float __k, float __u)
- long double [__gnu_cxx::jacobi_dnl](#) (long double __k, long double __u)
- template<typename _Kp, typename _Up >
[__gnu_cxx::__promote_fp_t](#)< _Kp, _Up > [__gnu_cxx::jacobi_sn](#) (_Kp __k, _Up __u)
- float [__gnu_cxx::jacobi_snf](#) (float __k, float __u)
- long double [__gnu_cxx::jacobi_snl](#) (long double __k, long double __u)
- template<typename _Tk, typename _Tphi >
[__gnu_cxx::__promote_fp_t](#)< _Tk, _Tphi > [__gnu_cxx::jacobi_zeta](#) (_Tk __k, _Tphi __phi)
- float [__gnu_cxx::jacobi_zetaf](#) (float __k, float __phi)
- long double [__gnu_cxx::jacobi_zetal](#) (long double __k, long double __phi)
- float [__gnu_cxx::jacobif](#) (unsigned __n, float __alpha, float __beta, float __x)
- long double [__gnu_cxx::jacobil](#) (unsigned __n, long double __alpha, long double __beta, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [std::laguerre](#) (unsigned int __n, _Tp __x)
- float [std::laguerref](#) (unsigned int __n, float __x)
- long double [std::laguerrel](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [__gnu_cxx::lbinomial](#) (unsigned int __n, unsigned int __k)

Return the logarithm of the binomial coefficient as a real number. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

- float `__gnu_cxx::lbinomialf` (unsigned int __n, unsigned int __k)
- long double `__gnu_cxx::lbinomiall` (unsigned int __n, unsigned int __k)
- template<typename _Tp >
`__gnu_cxx::__promote_fp_t<_Tp>` `__gnu_cxx::ldouble_factorial` (int __n)

Return the logarithm of the double factorial $\ln(n!!)$ of the argument as a real number.

$$n!! = n(n-2)\dots(2), 0!! = 1$$

for even n and

$$n!! = n(n-2)\dots(1), (-1)!! = 1$$

for odd n .

- float `__gnu_cxx::ldouble_factorialf` (int __n)
- long double `__gnu_cxx::ldouble_factoriall` (int __n)
- template<typename _Tp >
`__gnu_cxx::__promote_fp_t<_Tp>` `std::legendre` (unsigned int __l, _Tp __x)
- template<typename _Tp >
`__gnu_cxx::__promote_fp_t<_Tp>` `__gnu_cxx::legendre_q` (unsigned int __l, _Tp __x)
- float `__gnu_cxx::legendre_qf` (unsigned int __l, float __x)
- long double `__gnu_cxx::legendre_ql` (unsigned int __l, long double __x)
- float `std::legendref` (unsigned int __l, float __x)
- long double `std::legendrel` (unsigned int __l, long double __x)
- template<typename _Tp >
`__gnu_cxx::__promote_fp_t<_Tp>` `__gnu_cxx::lfactorial` (unsigned int __n)

Return the logarithm of the factorial $\ln(n!)$ of the argument as a real number.

$$n! = 1 \times 2 \times \dots \times n, 0! = 1$$

- float `__gnu_cxx::lfactorialf` (unsigned int __n)
- long double `__gnu_cxx::lfactoriall` (unsigned int __n)
- template<typename _Tp, typename _Tnu >
`__gnu_cxx::__promote_fp_t<_Tp, _Tnu>` `__gnu_cxx::lfalling_factorial` (_Tp __a, _Tnu __nu)

Return the logarithm of the falling factorial function or the lower Pochhammer symbol. The falling factorial function is defined by

$$a^{\underline{n}} = \Gamma(a+1)/\Gamma(a-n+1) = \prod_{k=0}^{n-1} (a-k), a^{\underline{0}} = 1$$

In particular, $f[n^{\underline{\{n\}}}] = n!$. Thus this function returns

$$\ln[a^{\underline{n}}] = \ln[\Gamma(a+1)] - \ln[\Gamma(a-n+1)], \ln[a^{\underline{0}}] = 0$$

Many notations exist for this function: $(a)_{\nu}$,

$$\left\{ \begin{matrix} a \\ \nu \end{matrix} \right\}$$

, and others.

- float `__gnu_cxx::lfalling_factorialf` (float __a, float __nu)
- long double `__gnu_cxx::lfalling_factoriall` (long double __a, long double __nu)

- `template<typename _Ta >`
`__gnu_cxx::__promote_fp_t< _Ta > __gnu_cxx::lgamma (_Ta __a)`
- `template<typename _Ta >`
`std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::lgamma (std::complex< _Ta > __a)`
- `float __gnu_cxx::lgammaf (float __a)`
- `std::complex< float > __gnu_cxx::lgammaf (std::complex< float > __a)`
- `long double __gnu_cxx::lgammal (long double __a)`
- `std::complex< long double > __gnu_cxx::lgammal (std::complex< long double > __a)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::logint (_Tp __x)`
- `float __gnu_cxx::logintf (float __x)`
- `long double __gnu_cxx::logintl (long double __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::logistic_cdf (_Ta __a, _Tb __b, _Tp __x)`
Return the logistic cumulative distribution function.
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::logistic_pdf (_Ta __a, _Tb __b, _Tp __x)`
Return the logistic probability density function.
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::lognormal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the lognormal cumulative probability density function.
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::lognormal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the lognormal probability density function.
- `template<typename _Tp, typename _Tnu >`
`__gnu_cxx::__promote_fp_t< _Tp, _Tnu > __gnu_cxx::lrising_factorial (_Tp __a, _Tnu __nu)`
Return the logarithm of the rising factorial function or the (upper) Pochhammer symbol. The rising factorial function is defined for integer order by

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(a) = \prod_{k=0}^{\nu-1} (a + k), \overline{0} = 1$$

Thus this function returns

$$\ln[a^{\overline{\nu}}] = \ln[\Gamma(a + \nu)] - \ln[\Gamma(a)], \ln[a^{\overline{0}}] = 0$$

Many notations exist for this function: $(a)_{\nu}$ (especially in the literature of special functions),

$$\left[\begin{matrix} a \\ \nu \end{matrix} \right]$$

, and others.

- `float __gnu_cxx::lrising_factorialf (float __a, float __nu)`
- `long double __gnu_cxx::lrising_factoriall (long double __a, long double __nu)`
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::normal_cdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the normal cumulative probability density function.
- `template<typename _Tmu, typename _Tsig, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tmu, _Tsig, _Tp > __gnu_cxx::normal_pdf (_Tmu __mu, _Tsig __sigma, _Tp __x)`
Return the normal probability density function.
- `template<typename _Tph, typename _Tpa >`
`__gnu_cxx::__promote_fp_t< _Tph, _Tpa > __gnu_cxx::owens_t (_Tph __h, _Tpa __a)`

- float [__gnu_cxx::owens_tf](#) (float __h, float __a)
- long double [__gnu_cxx::owens_tl](#) (long double __h, long double __a)
- template<typename _Ta, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Ta, _Tp>](#) [__gnu_cxx::pgamma](#) (_Ta __a, _Tp __x)
- float [__gnu_cxx::pgammaf](#) (float __a, float __x)
- long double [__gnu_cxx::pgammal](#) (long double __a, long double __x)
- template<typename _Tp, typename _Wp >
[__gnu_cxx::__promote_fp_t<_Tp, _Wp>](#) [__gnu_cxx::polylog](#) (_Tp __s, _Wp __w)
- template<typename _Tp, typename _Wp >
std::complex< [__gnu_cxx::__promote_fp_t<_Tp, _Wp>](#) > [__gnu_cxx::polylog](#) (_Tp __s, std::complex< _Tp > __w)
- float [__gnu_cxx::polylogf](#) (float __s, float __w)
- std::complex< float > [__gnu_cxx::polylogf](#) (float __s, std::complex< float > __w)
- long double [__gnu_cxx::polylogl](#) (long double __s, long double __w)
- std::complex< long double > [__gnu_cxx::polylogl](#) (long double __s, std::complex< long double > __w)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::psi](#) (_Tp __x)
- float [__gnu_cxx::psif](#) (float __x)
- long double [__gnu_cxx::psil](#) (long double __x)
- template<typename _Ta, typename _Tp >
[__gnu_cxx::__promote_fp_t<_Ta, _Tp>](#) [__gnu_cxx::qgamma](#) (_Ta __a, _Tp __x)
- float [__gnu_cxx::qgammaf](#) (float __a, float __x)
- long double [__gnu_cxx::qgammal](#) (long double __a, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::radpoly](#) (unsigned int __n, unsigned int __m, _Tp __rho)
- float [__gnu_cxx::radpolyf](#) (unsigned int __n, unsigned int __m, float __rho)
- long double [__gnu_cxx::radpolyl](#) (unsigned int __n, unsigned int __m, long double __rho)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [std::riemann_zeta](#) (_Tp __s)
- float [std::riemann_zetaf](#) (float __s)
- long double [std::riemann_zetal](#) (long double __s)
- template<typename _Tp, typename _Tnu >
[__gnu_cxx::__promote_fp_t<_Tp, _Tnu>](#) [__gnu_cxx::rising_factorial](#) (_Tp __a, _Tnu __nu)

Return the rising factorial function or the (upper) Pochhammer function. The rising factorial function is defined by

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(\nu)$$

Many notations exist for this function: $(a)_{\nu}$, (especially in the literature of special functions),

$$\begin{bmatrix} a \\ n \end{bmatrix}$$

, and others.

- float [__gnu_cxx::rising_factorialf](#) (float __a, float __nu)
- long double [__gnu_cxx::rising_factoriall](#) (long double __a, long double __nu)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::sin_pi](#) (_Tp __x)
- float [__gnu_cxx::sin_pif](#) (float __x)
- long double [__gnu_cxx::sin_pil](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::sinc](#) (_Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t<_Tp>](#) [__gnu_cxx::sinc_pi](#) (_Tp __x)
- float [__gnu_cxx::sinc_pif](#) (float __x)

- long double [__gnu_cxx::sinc_pil](#) (long double __x)
- float [__gnu_cxx::sincf](#) (float __x)
- long double [__gnu_cxx::sincl](#) (long double __x)
- [__gnu_cxx::__sincos_t](#)< double > [__gnu_cxx::sincos](#) (double __x)
- template<typename _Tp >
[__gnu_cxx::__sincos_t](#)< [__gnu_cxx::__promote_fp_t](#)< _Tp > > [__gnu_cxx::sincos](#) (_Tp __x)
- template<typename _Tp >
[__gnu_cxx::__sincos_t](#)< [__gnu_cxx::__promote_fp_t](#)< _Tp > > [__gnu_cxx::sincos_pi](#) (_Tp __x)
- [__gnu_cxx::__sincos_t](#)< float > [__gnu_cxx::sincos_pif](#) (float __x)
- [__gnu_cxx::__sincos_t](#)< long double > [__gnu_cxx::sincos_pil](#) (long double __x)
- [__gnu_cxx::__sincos_t](#)< float > [__gnu_cxx::sincosf](#) (float __x)
- [__gnu_cxx::__sincos_t](#)< long double > [__gnu_cxx::sincosl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [__gnu_cxx::sinh_pi](#) (_Tp __x)
- float [__gnu_cxx::sinh_pif](#) (float __x)
- long double [__gnu_cxx::sinh_pil](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [__gnu_cxx::sinhc](#) (_Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [__gnu_cxx::sinhc_pi](#) (_Tp __x)
- float [__gnu_cxx::sinhc_pif](#) (float __x)
- long double [__gnu_cxx::sinhc_pil](#) (long double __x)
- float [__gnu_cxx::sinhcf](#) (float __x)
- long double [__gnu_cxx::sinhcl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [__gnu_cxx::sinhint](#) (_Tp __x)
- float [__gnu_cxx::sinhintf](#) (float __x)
- long double [__gnu_cxx::sinhintl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [__gnu_cxx::sinint](#) (_Tp __x)
- float [__gnu_cxx::sinintf](#) (float __x)
- long double [__gnu_cxx::sinintl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [std::sph_bessel](#) (unsigned int __n, _Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [__gnu_cxx::sph_bessel_i](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::sph_bessel_if](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::sph_bessel_il](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_fp_t](#)< _Tp > [__gnu_cxx::sph_bessel_k](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::sph_bessel_kf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::sph_bessel_kl](#) (unsigned int __n, long double __x)
- float [std::sph_besself](#) (unsigned int __n, float __x)
- long double [std::sph_bessell](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[std::complex](#)< [__gnu_cxx::__promote_fp_t](#)< _Tp > > [__gnu_cxx::sph_hankel_1](#) (unsigned int __n, _Tp __z)
- template<typename _Tp >
[std::complex](#)< [__gnu_cxx::__promote_fp_t](#)< _Tp > > [__gnu_cxx::sph_hankel_1](#) (unsigned int __n, [std::complex](#)< _Tp > __x)
- [std::complex](#)< float > [__gnu_cxx::sph_hankel_1f](#) (unsigned int __n, float __z)
- [std::complex](#)< float > [__gnu_cxx::sph_hankel_1f](#) (unsigned int __n, [std::complex](#)< float > __x)

- `std::complex< long double > __gnu_cxx::sph_hankel_1l` (unsigned int __n, long double __z)
- `std::complex< long double > __gnu_cxx::sph_hankel_1l` (unsigned int __n, `std::complex< long double > __x`)
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sph_hankel_2` (unsigned int __n, _Tp __z)
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_fp_t< _Tp > > __gnu_cxx::sph_hankel_2` (unsigned int __n, `std::complex< _Tp > __x`)
- `std::complex< float > __gnu_cxx::sph_hankel_2f` (unsigned int __n, float __z)
- `std::complex< float > __gnu_cxx::sph_hankel_2f` (unsigned int __n, `std::complex< float > __x`)
- `std::complex< long double > __gnu_cxx::sph_hankel_2l` (unsigned int __n, long double __z)
- `std::complex< long double > __gnu_cxx::sph_hankel_2l` (unsigned int __n, `std::complex< long double > __x`)
- `template<typename _Ttheta, typename _Tphi >`
`std::complex< __gnu_cxx::__promote_fp_t< _Ttheta, _Tphi > > __gnu_cxx::sph_harmonic` (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)
- `std::complex< float > __gnu_cxx::sph_harmonicf` (unsigned int __l, int __m, float __theta, float __phi)
- `std::complex< long double > __gnu_cxx::sph_harmonicl` (unsigned int __l, int __m, long double __theta, long double __phi)
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::sph_legendre` (unsigned int __l, unsigned int __m, _Tp __theta)
- `float std::sph_legendref` (unsigned int __l, unsigned int __m, float __theta)
- `long double std::sph_legendrel` (unsigned int __l, unsigned int __m, long double __theta)
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > std::sph_neumann` (unsigned int __n, _Tp __x)
- `float std::sph_neumannf` (unsigned int __n, float __x)
- `long double std::sph_neumannl` (unsigned int __n, long double __x)
- `template<typename _Tp >`
`_Tp __gnu_cxx::stirling_1` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp __gnu_cxx::stirling_2` (unsigned int __n, unsigned int __m)
- `template<typename _Tt, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::student_t_cdf` (_Tt __t, unsigned int __nu)
Return the Students T probability function.
- `template<typename _Tt, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::student_t_pdf` (_Tt __t, unsigned int __nu)
Return the complement of the Students T probability function.
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::tan_pi` (_Tp __x)
- `float __gnu_cxx::tan_pif` (float __x)
- `long double __gnu_cxx::tan_pil` (long double __x)
- `template<typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Tp > __gnu_cxx::tanh_pi` (_Tp __x)
- `float __gnu_cxx::tanh_pif` (float __x)
- `long double __gnu_cxx::tanh_pil` (long double __x)
- `template<typename _Ta >`
`__gnu_cxx::__promote_fp_t< _Ta > __gnu_cxx::tgamma` (_Ta __a)
- `template<typename _Ta >`
`std::complex< __gnu_cxx::__promote_fp_t< _Ta > > __gnu_cxx::tgamma` (`std::complex< _Ta > __a`)
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma` (_Ta __a, _Tp __x)
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::__promote_fp_t< _Ta, _Tp > __gnu_cxx::tgamma_lower` (_Ta __a, _Tp __x)

- float [__gnu_cxx::tgamma_lowerf](#) (float __a, float __x)
- long double [__gnu_cxx::tgamma_lowerl](#) (long double __a, long double __x)
- float [__gnu_cxx::tgammaf](#) (float __a)
- std::complex< float > [__gnu_cxx::tgammaf](#) (std::complex< float > __a)
- float [__gnu_cxx::tgammaf](#) (float __a, float __x)
- long double [__gnu_cxx::tgammal](#) (long double __a)
- std::complex< long double > [__gnu_cxx::tgammal](#) (std::complex< long double > __a)
- long double [__gnu_cxx::tgammal](#) (long double __a, long double __x)
- template<typename _Tpnu, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_1](#) (_Tpnu __nu, _Tp __x)
- float [__gnu_cxx::theta_1f](#) (float __nu, float __x)
- long double [__gnu_cxx::theta_1l](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_2](#) (_Tpnu __nu, _Tp __x)
- float [__gnu_cxx::theta_2f](#) (float __nu, float __x)
- long double [__gnu_cxx::theta_2l](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_3](#) (_Tpnu __nu, _Tp __x)
- float [__gnu_cxx::theta_3f](#) (float __nu, float __x)
- long double [__gnu_cxx::theta_3l](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Tpnu, _Tp > __gnu_cxx::theta_4](#) (_Tpnu __nu, _Tp __x)
- float [__gnu_cxx::theta_4f](#) (float __nu, float __x)
- long double [__gnu_cxx::theta_4l](#) (long double __nu, long double __x)
- template<typename _Tp_k, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Tp_k, _Tp > __gnu_cxx::theta_c](#) (_Tp_k __k, _Tp __x)
- float [__gnu_cxx::theta_cf](#) (float __k, float __x)
- long double [__gnu_cxx::theta_cl](#) (long double __k, long double __x)
- template<typename _Tp_k, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Tp_k, _Tp > __gnu_cxx::theta_d](#) (_Tp_k __k, _Tp __x)
- float [__gnu_cxx::theta_df](#) (float __k, float __x)
- long double [__gnu_cxx::theta_dl](#) (long double __k, long double __x)
- template<typename _Tp_k, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Tp_k, _Tp > __gnu_cxx::theta_n](#) (_Tp_k __k, _Tp __x)
- float [__gnu_cxx::theta_nf](#) (float __k, float __x)
- long double [__gnu_cxx::theta_nl](#) (long double __k, long double __x)
- template<typename _Tp_k, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Tp_k, _Tp > __gnu_cxx::theta_s](#) (_Tp_k __k, _Tp __x)
- float [__gnu_cxx::theta_sf](#) (float __k, float __x)
- long double [__gnu_cxx::theta_sl](#) (long double __k, long double __x)
- template<typename _Tpa, typename _Tpc, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Tpa, _Tpc, _Tp > __gnu_cxx::tricomi_u](#) (_Tpa __a, _Tpc __c, _Tp __x)
- float [__gnu_cxx::tricomi_uf](#) (float __a, float __c, float __x)
- long double [__gnu_cxx::tricomi_ul](#) (long double __a, long double __c, long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::weibull_cdf](#) (_Ta __a, _Tb __b, _Tp __x)
Return the Weibull cumulative probability density function.
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::__promote_fp_t< _Ta, _Tb, _Tp > __gnu_cxx::weibull_pdf](#) (_Ta __a, _Tb __b, _Tp __x)
Return the Weibull probability density function.

- `template<typename _Trho, typename _Tphi >
__gnu_cxx::__promote_fp_t< _Trho, _Tphi > __gnu_cxx::zernike` (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)
- `float __gnu_cxx::zernikef` (unsigned int __n, int __m, float __rho, float __phi)
- `long double __gnu_cxx::zernikel` (unsigned int __n, int __m, long double __rho, long double __phi)

11.31.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.31.2 Macro Definition Documentation

11.31.2.1 `#define __cpp_lib_math_special_functions 201603L`

Definition at line 39 of file `specfun.h`.

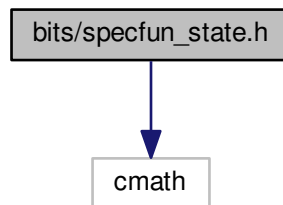
11.31.2.2 `#define __STDCPP_MATH_SPEC_FUNCS__ 201003L`

Definition at line 37 of file `specfun.h`.

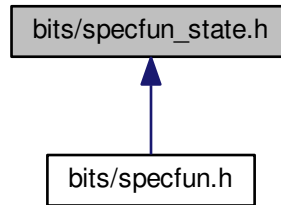
11.32 bits/specfun_state.h File Reference

```
#include <cmath>
```

Include dependency graph for `specfun_state.h`:



This graph shows which files directly or indirectly include this file:



Classes

- struct [__gnu_cxx::__airy_t](#)< _Tx, _Tp >
- struct [__gnu_cxx::__cyl_bessel_t](#)< _Tnu, _Tx, _Tp >
- struct [__gnu_cxx::__cyl_hankel_t](#)< _Tnu, _Tx, _Tp >
- struct [__gnu_cxx::__cyl_mod_bessel_t](#)< _Tnu, _Tx, _Tp >
- struct [__gnu_cxx::__fock_airy_t](#)< _Tx, _Tp >
- struct [__gnu_cxx::__gamma_inc_t](#)< _Tp >
- struct [__gnu_cxx::__gamma_temme_t](#)< _Tp >

A structure for the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ 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.

- struct [__gnu_cxx::__jacobi_t](#)< _Tp >
- struct [__gnu_cxx::__lgamma_t](#)< _Tp >
- struct [__gnu_cxx::__pqgamma_t](#)< _Tp >
- struct [__gnu_cxx::__quadrature_point_t](#)< _Tp >
- struct [__gnu_cxx::__sincos_t](#)< _Tp >
- struct [__gnu_cxx::__sph_bessel_t](#)< _Tn, _Tx, _Tp >
- struct [__gnu_cxx::__sph_hankel_t](#)< _Tn, _Tx, _Tp >
- struct [__gnu_cxx::__sph_mod_bessel_t](#)< _Tn, _Tx, _Tp >

Namespaces

- [__gnu_cxx](#)

11.32.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.33 ext/math_util.h File Reference

Classes

- struct [__gnu_cxx::__fp_is_integer_t](#)

Namespaces

- [__gnu_cxx](#)

Functions

- `template<typename _Tp >`
`bool __gnu_cxx::__fp_is_equal (_Tp __a, _Tp __b, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`__fp_is_integer_t __gnu_cxx::__fp_is_even_integer (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`__fp_is_integer_t __gnu_cxx::__fp_is_half_integer (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`__fp_is_integer_t __gnu_cxx::__fp_is_half_odd_integer (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`__fp_is_integer_t __gnu_cxx::__fp_is_integer (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`__fp_is_integer_t __gnu_cxx::__fp_is_odd_integer (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`bool __gnu_cxx::__fp_is_zero (_Tp __a, _Tp __mul=_Tp{1})`
- `template<typename _Tp >`
`_Tp __gnu_cxx::__fp_max_abs (_Tp __a, _Tp __b)`
- `template<typename _Tp, typename _IntTp >`
`_Tp __gnu_cxx::__parity (_IntTp __k)`

11.33.1 Detailed Description

This file is a GNU extension to the Standard C++ Library.

Index

- `_Airy`
 - `std::__detail::_Airy`, [365](#)
- `_Airy_asymp`
 - `std::__detail::_Airy_asymp`, [368](#)
- `_Airy_asymp_series`
 - `std::__detail::_Airy_asymp_series`, [374](#)
- `_AsympTerminator`
 - `std::__detail::_AsympTerminator`, [388](#)
- `_Cmplx`
 - `std::__detail::_Airy_asymp`, [368](#)
 - `std::__detail::_Airy_series`, [378](#)
- `_GLIBCXX_BITS_SF_AIRY_TCC`
 - `sf_airy.tcc`, [393](#)
- `_GLIBCXX_BITS_SF_BERNOULLI_TCC`
 - `sf_bernoulli.tcc`, [394](#)
- `_GLIBCXX_BITS_SF_BESSEL_TCC`
 - `sf_bessel.tcc`, [397](#)
- `_GLIBCXX_BITS_SF_BETA_TCC`
 - `sf_beta.tcc`, [398](#)
- `_GLIBCXX_BITS_SF_CARDINAL_TCC`
 - `sf_cardinal.tcc`, [400](#)
- `_GLIBCXX_BITS_SF_CHEBYSHEV_TCC`
 - `sf_chebyshev.tcc`, [402](#)
- `_GLIBCXX_BITS_SF_DAWSON_TCC`
 - `sf_dawson.tcc`, [403](#)
- `_GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC`
 - `sf_distributions.tcc`, [406](#)
- `_GLIBCXX_BITS_SF_ELLINT_TCC`
 - `sf_ellint.tcc`, [408](#)
- `_GLIBCXX_BITS_SF_EULER_TCC`
 - `sf_euler.tcc`, [410](#)
- `_GLIBCXX_BITS_SF_EXPINT_TCC`
 - `sf_expint.tcc`, [412](#)
- `_GLIBCXX_BITS_SF_FRESNEL_TCC`
 - `sf_fresnel.tcc`, [414](#)
- `_GLIBCXX_BITS_SF_GAMMA_TCC`
 - `sf_gamma.tcc`, [422](#)
- `_GLIBCXX_BITS_SF_GEGENBAUER_TCC`
 - `sf_gegenbauer.tcc`, [423](#)
- `_GLIBCXX_BITS_SF_HANKEL_TCC`
 - `sf_hankel.tcc`, [426](#)
- `_GLIBCXX_BITS_SF_HERMITE_TCC`
 - `sf_hermite.tcc`, [428](#)
- `_GLIBCXX_BITS_SF_HYDROGEN_TCC`
 - `sf_hydrogen.tcc`, [429](#)
- `_GLIBCXX_BITS_SF_HYPERG_TCC`
 - `sf_hyperg.tcc`, [431](#)
- `_GLIBCXX_BITS_SF_HYPINT_TCC`
 - `sf_hypint.tcc`, [433](#)
- `_GLIBCXX_BITS_SF_JACOBI_TCC`
 - `sf_jacobi.tcc`, [434](#)
- `_GLIBCXX_BITS_SF_LAGUERRE_TCC`
 - `sf_laguerre.tcc`, [436](#)
- `_GLIBCXX_BITS_SF_LEGENDRE_TCC`
 - `sf_legendre.tcc`, [438](#)
- `_GLIBCXX_BITS_SF_MOD_BESSEL_TCC`
 - `sf_mod_bessel.tcc`, [440](#)
- `_GLIBCXX_BITS_SF_OWENS_T_TCC`
 - `sf_owens_t.tcc`, [441](#)
- `_GLIBCXX_BITS_SF_POLYLOG_TCC`
 - `sf_polylog.tcc`, [444](#)
- `_GLIBCXX_BITS_SF_STIRLING_TCC`
 - `sf_stirling.tcc`, [445](#)
- `_GLIBCXX_BITS_SF_THETA_TCC`
 - `sf_theta.tcc`, [447](#)
- `_GLIBCXX_BITS_SF_TRIGINT_TCC`
 - `sf_trigint.tcc`, [451](#)
- `_GLIBCXX_BITS_SF_TRIG_TCC`
 - `sf_trig.tcc`, [449](#)
- `_GLIBCXX_BITS_SF_ZETA_TCC`
 - `sf_zeta.tcc`, [453](#)
- `_N_FGH`
 - `std::__detail::_Airy_series`, [382](#)
- `_Num_Euler_Maclaurin_zeta`
 - `std::__detail`, [329](#)
- `_Real`
 - `std::__detail::_AiryState`, [386](#)
- `_S_Ai`
 - `std::__detail::_Airy_series`, [379](#)
- `_S_Ai0`
 - `std::__detail::_Airy_series`, [382](#)
- `_S_Aip0`
 - `std::__detail::_Airy_series`, [382](#)
- `_S_Airy`
 - `std::__detail::_Airy_series`, [379](#)
- `_S_Bi`
 - `std::__detail::_Airy_series`, [379](#)
- `_S_Bi0`
 - `std::__detail::_Airy_series`, [382](#)
- `_S_Bip0`

- std::__detail::Airy_series, 382
- _S_Euler_Maclaurin_zeta
 - std::__detail, 330
- _S_FGH
 - std::__detail::Airy_series, 380
- _S_Fock
 - std::__detail::Airy_series, 380
- _S_Gi0
 - std::__detail::Airy_series, 383
- _S_Gip0
 - std::__detail::Airy_series, 383
- _S_Hi0
 - std::__detail::Airy_series, 383
- _S_Hip0
 - std::__detail::Airy_series, 383
- _S_Scorer
 - std::__detail::Airy_series, 381
- _S_Scorer2
 - std::__detail::Airy_series, 381
- _S_absarg_ge_pio3
 - std::__detail::Airy_asymp, 368
- _S_absarg_lt_pio3
 - std::__detail::Airy_asymp, 368
- _S_c
 - std::__detail::Airy_asymp_data< double >, 371
 - std::__detail::Airy_asymp_data< float >, 372
 - std::__detail::Airy_asymp_data< long double >, 372
- _S_cheby
 - std::__detail::gamma_lanczos_data< double >, 359
 - std::__detail::gamma_lanczos_data< float >, 360
 - std::__detail::gamma_lanczos_data< long double >, 361
 - std::__detail::gamma_spouge_data< double >, 362
 - std::__detail::gamma_spouge_data< float >, 363
 - std::__detail::gamma_spouge_data< long double >, 364
- _S_d
 - std::__detail::Airy_asymp_data< double >, 371
 - std::__detail::Airy_asymp_data< float >, 372
 - std::__detail::Airy_asymp_data< long double >, 372
- _S_double_factorial_table
 - std::__detail, 330
- _S_eps
 - std::__detail::Airy_series, 383
- _S_factorial_table
 - std::__detail, 330
- _S_g
 - std::__detail::gamma_lanczos_data< double >, 359
 - std::__detail::gamma_lanczos_data< float >, 360
- std::__detail::gamma_lanczos_data< long double >, 361
- _S_harmonic_denom
 - std::__detail, 330
- _S_harmonic_numer
 - std::__detail, 330
- _S_i
 - std::__detail::Airy_series, 383
- _S_max_cd
 - std::__detail::Airy_asymp_data< double >, 371
 - std::__detail::Airy_asymp_data< float >, 372
 - std::__detail::Airy_asymp_data< long double >, 373
- _S_neg_double_factorial_table
 - std::__detail, 330
- _S_num_double_factorials
 - std::__detail, 331
- _S_num_double_factorials< double >
 - std::__detail, 331
- _S_num_double_factorials< float >
 - std::__detail, 331
- _S_num_double_factorials< long double >
 - std::__detail, 331
- _S_num_factorials
 - std::__detail, 331
- _S_num_factorials< double >
 - std::__detail, 331
- _S_num_factorials< float >
 - std::__detail, 331
- _S_num_factorials< long double >
 - std::__detail, 331
- _S_num_harmonic_numer
 - std::__detail, 331
- _S_num_neg_double_factorials
 - std::__detail, 332
- _S_num_neg_double_factorials< double >
 - std::__detail, 332
- _S_num_neg_double_factorials< float >
 - std::__detail, 332
- _S_num_neg_double_factorials< long double >
 - std::__detail, 332
- _S_num_zetam1
 - std::__detail, 332
- _S_pi
 - std::__detail::Airy_series, 383
- _S_sqrt_pi
 - std::__detail::Airy_asymp_series, 375
 - std::__detail::Airy_series, 383
- _S_zetam1
 - std::__detail, 332
- _Terminator
 - std::__detail::Terminator, 390
- _Val
 - std::__detail::AiryAuxilliaryState, 384

- __Ai_deriv
 - __gnu_cxx::__airy_t, 334
 - std::__detail::AiryState, 387
- __Ai_value
 - __gnu_cxx::__airy_t, 334
 - std::__detail::AiryState, 387
- __Bi_deriv
 - __gnu_cxx::__airy_t, 334
 - std::__detail::AiryState, 387
- __Bi_value
 - __gnu_cxx::__airy_t, 334
 - std::__detail::AiryState, 387
- __H1_deriv
 - __gnu_cxx::__cyl_hankel_t, 338
- __H1_value
 - __gnu_cxx::__cyl_hankel_t, 338
- __H2_deriv
 - __gnu_cxx::__cyl_hankel_t, 338
- __H2_value
 - __gnu_cxx::__cyl_hankel_t, 338
- __I_deriv
 - __gnu_cxx::__cyl_mod_bessel_t, 340
- __I_value
 - __gnu_cxx::__cyl_mod_bessel_t, 340
- __J_deriv
 - __gnu_cxx::__cyl_bessel_t, 336
- __J_value
 - __gnu_cxx::__cyl_bessel_t, 336
- __K_deriv
 - __gnu_cxx::__cyl_mod_bessel_t, 340
- __K_value
 - __gnu_cxx::__cyl_mod_bessel_t, 340
- __N_deriv
 - __gnu_cxx::__cyl_bessel_t, 336
- __N_value
 - __gnu_cxx::__cyl_bessel_t, 336
- __STDCPP_MATH_SPEC_FUNCS__
 - specfun.h, 469
- __Wronskian
 - __gnu_cxx::__airy_t, 334
 - __gnu_cxx::__cyl_bessel_t, 335
 - __gnu_cxx::__cyl_hankel_t, 337
 - __gnu_cxx::__cyl_mod_bessel_t, 339
 - __gnu_cxx::__fock_airy_t, 341
 - __gnu_cxx::__sph_bessel_t, 353
 - __gnu_cxx::__sph_hankel_t, 355
 - __gnu_cxx::__sph_mod_bessel_t, 357
- __airy
 - std::__detail, 206
- __airy_ai
 - std::__detail, 206
- __airy_arg
 - std::__detail, 206
- __airy_bi
 - std::__detail, 207
- __am
 - __gnu_cxx::__jacobi_t, 347
- __assoc_laguerre
 - std::__detail, 207
- __assoc_legendre_p
 - std::__detail, 208
- __bernoulli
 - std::__detail, 208, 209
- __bernoulli_2n
 - std::__detail, 209
- __bernoulli_series
 - std::__detail, 210
- __beta
 - std::__detail, 210
- __beta_gamma
 - std::__detail, 211
- __beta_inc
 - std::__detail, 211
- __beta_lgamma
 - std::__detail, 212
- __beta_product
 - std::__detail, 212
- __binomial
 - std::__detail, 213
- __binomial_cdf
 - std::__detail, 214
- __binomial_cdfc
 - std::__detail, 214
- __binomial_pdf
 - std::__detail, 215
- __bose_einstein
 - std::__detail, 215
- __cd
 - __gnu_cxx::__jacobi_t, 347
- __chebyshev_recur
 - std::__detail, 216
- __chebyshev_t
 - std::__detail, 216
- __chebyshev_u
 - std::__detail, 217
- __chebyshev_v
 - std::__detail, 218
- __chebyshev_w
 - std::__detail, 218
- __chi_squared_pdf
 - std::__detail, 219
- __chi_squared_pdfc
 - std::__detail, 219
- __chshint
 - std::__detail, 219
- __chshint_cont_frac
 - std::__detail, 219
- __chshint_series

std::__detail, 220
 __clamp_0_m2pi
 std::__detail, 220
 __clamp_pi
 std::__detail, 220
 __clausen
 std::__detail, 220, 221
 __clausen_cl
 std::__detail, 221, 222
 __clausen_sl
 std::__detail, 222, 223
 __cn_value
 __gnu_cxx::__jacobi_t, 348
 __comp_ellint_1
 std::__detail, 223
 __comp_ellint_2
 std::__detail, 224
 __comp_ellint_3
 std::__detail, 224
 __comp_ellint_d
 std::__detail, 225
 __comp_ellint_rf
 std::__detail, 225
 __comp_ellint_rg
 std::__detail, 225
 __conf_hyperg
 std::__detail, 225
 __conf_hyperg_lim
 std::__detail, 226
 __conf_hyperg_lim_series
 std::__detail, 226
 __conf_hyperg_luke
 std::__detail, 227
 __conf_hyperg_series
 std::__detail, 227
 __cos_pi
 std::__detail, 227, 228
 __cos_v
 __gnu_cxx::__sincos_t, 352
 __cosh_pi
 std::__detail, 228
 __coshint
 std::__detail, 228
 __cpp_lib_math_special_functions
 specfun.h, 469
 __cs
 __gnu_cxx::__jacobi_t, 347
 __cyl_bessel
 std::__detail, 229
 __cyl_bessel_i
 std::__detail, 229
 __cyl_bessel_ij_series
 std::__detail, 230
 __cyl_bessel_ik
 std::__detail, 230
 __cyl_bessel_ik_asymp
 std::__detail, 231
 __cyl_bessel_ik_steep
 std::__detail, 231
 __cyl_bessel_j
 std::__detail, 232
 __cyl_bessel_jn
 std::__detail, 232
 __cyl_bessel_jn_asymp
 std::__detail, 232
 __cyl_bessel_jn_neg_arg
 std::__detail, 233
 __cyl_bessel_jn_steep
 std::__detail, 233
 __cyl_bessel_k
 std::__detail, 233
 __cyl_hankel_1
 std::__detail, 234
 __cyl_hankel_2
 std::__detail, 235
 __cyl_neumann
 std::__detail, 236
 __cyl_neumann_n
 std::__detail, 236
 __dawson
 std::__detail, 237
 __dawson_cont_frac
 std::__detail, 237
 __dawson_series
 std::__detail, 237
 __dc
 __gnu_cxx::__jacobi_t, 347
 __debye
 std::__detail, 237
 __debye_region
 std::__detail, 238
 __dilog
 std::__detail, 238
 __dirichlet_beta
 std::__detail, 239
 __dirichlet_eta
 std::__detail, 240
 __dirichlet_lambda
 std::__detail, 241
 __dn_value
 __gnu_cxx::__jacobi_t, 348
 __double_factorial
 std::__detail, 241
 __ds
 __gnu_cxx::__jacobi_t, 348
 __ellint_1
 std::__detail, 242
 __ellint_2

- `std::__detail`, [242](#)
- `__ellint_3`
 - `std::__detail`, [243](#)
- `__ellint_cel`
 - `std::__detail`, [243](#)
- `__ellint_d`
 - `std::__detail`, [243](#)
- `__ellint_el1`
 - `std::__detail`, [244](#)
- `__ellint_el2`
 - `std::__detail`, [244](#)
- `__ellint_el3`
 - `std::__detail`, [244](#)
- `__ellint_rc`
 - `std::__detail`, [244](#)
- `__ellint_rd`
 - `std::__detail`, [245](#)
- `__ellint_rf`
 - `std::__detail`, [246](#)
- `__ellint_rg`
 - `std::__detail`, [246](#)
- `__ellint_rj`
 - `std::__detail`, [247](#)
- `__ellnome`
 - `std::__detail`, [248](#)
- `__ellnome_k`
 - `std::__detail`, [248](#)
- `__ellnome_series`
 - `std::__detail`, [248](#)
- `__euler`
 - `std::__detail`, [248](#), [249](#)
- `__euler_series`
 - `std::__detail`, [249](#)
- `__eulerian_1_recur`
 - `std::__detail`, [249](#)
- `__eulerian_2`
 - `std::__detail`, [249](#)
- `__eulerian_2_recur`
 - `std::__detail`, [250](#)
- `__expint`
 - `std::__detail`, [250](#)
- `__expint_E1`
 - `std::__detail`, [251](#)
- `__expint_E1_asymp`
 - `std::__detail`, [251](#)
- `__expint_E1_series`
 - `std::__detail`, [252](#)
- `__expint_Ei`
 - `std::__detail`, [252](#)
- `__expint_Ei_asymp`
 - `std::__detail`, [253](#)
- `__expint_Ei_series`
 - `std::__detail`, [253](#)
- `__expint_En_asymp`
 - `std::__detail`, [254](#)
- `__expint_En_cont_frac`
 - `std::__detail`, [254](#)
- `__expint_En_large_n`
 - `std::__detail`, [255](#)
- `__expint_En_recursion`
 - `std::__detail`, [255](#)
- `__expint_En_series`
 - `std::__detail`, [256](#)
- `__exponential_cdf`
 - `std::__detail`, [256](#)
- `__exponential_cdfc`
 - `std::__detail`, [257](#)
- `__exponential_pdf`
 - `std::__detail`, [257](#)
- `__factorial`
 - `std::__detail`, [257](#)
 - `std::__detail::Factorial_table`, [389](#)
- `__fai_deriv`
 - `std::__detail::AiryAuxilliaryState`, [385](#)
- `__fai_value`
 - `std::__detail::AiryAuxilliaryState`, [385](#)
- `__falling_factorial`
 - `std::__detail`, [257](#), [258](#)
- `__fermi_dirac`
 - `std::__detail`, [258](#)
- `__fisher_f_cdf`
 - `std::__detail`, [259](#)
- `__fisher_f_cdfc`
 - `std::__detail`, [259](#)
- `__fisher_f_pdf`
 - `std::__detail`, [259](#)
- `__fock_airy`
 - `std::__detail`, [260](#)
- `__fp_is_equal`
 - `__gnu_cxx`, [178](#)
- `__fp_is_even_integer`
 - `__gnu_cxx`, [178](#)
- `__fp_is_half_integer`
 - `__gnu_cxx`, [179](#)
- `__fp_is_half_odd_integer`
 - `__gnu_cxx`, [179](#)
- `__fp_is_integer`
 - `__gnu_cxx`, [180](#)
- `__fp_is_odd_integer`
 - `__gnu_cxx`, [180](#)
- `__fp_is_zero`
 - `__gnu_cxx`, [181](#)
- `__fp_max_abs`
 - `__gnu_cxx`, [181](#)
- `__fresnel`
 - `std::__detail`, [260](#)
- `__fresnel_cont_frac`
 - `std::__detail`, [261](#)

- __fresnel_series
 - std::__detail, [261](#)
- __gai_deriv
 - std::__detail::__AiryAuxilliaryState, [385](#)
- __gai_value
 - std::__detail::__AiryAuxilliaryState, [385](#)
- __gamma
 - std::__detail, [261](#), [262](#)
- __gamma_1_value
 - __gnu_cxx::__gamma_temme_t, [346](#)
- __gamma_2_value
 - __gnu_cxx::__gamma_temme_t, [346](#)
- __gamma_cdf
 - std::__detail, [262](#)
- __gamma_cdfc
 - std::__detail, [262](#)
- __gamma_cont_frac
 - std::__detail, [262](#)
- __gamma_minus_value
 - __gnu_cxx::__gamma_temme_t, [346](#)
- __gamma_pdf
 - std::__detail, [263](#)
- __gamma_plus_value
 - __gnu_cxx::__gamma_temme_t, [346](#)
- __gamma_reciprocal
 - std::__detail, [263](#)
- __gamma_reciprocal_series
 - std::__detail, [263](#)
- __gamma_series
 - std::__detail, [264](#)
- __gamma_temme
 - std::__detail, [264](#)
- __gauss
 - std::__detail, [266](#)
- __gegenbauer_poly
 - std::__detail, [266](#)
- __gnu_cxx, [165](#)
 - __fp_is_equal, [178](#)
 - __fp_is_even_integer, [178](#)
 - __fp_is_half_integer, [179](#)
 - __fp_is_half_odd_integer, [179](#)
 - __fp_is_integer, [180](#)
 - __fp_is_odd_integer, [180](#)
 - __fp_is_zero, [181](#)
 - __fp_max_abs, [181](#)
 - __parity, [181](#)
- __gnu_cxx::__airy_t
 - __Ai_deriv, [334](#)
 - __Ai_value, [334](#)
 - __Bi_deriv, [334](#)
 - __Bi_value, [334](#)
 - __Wronskian, [334](#)
 - __x_arg, [334](#)
- __gnu_cxx::__airy_t<_Tx, _Tp>, [333](#)
- __gnu_cxx::__cyl_bessel_t
 - __J_deriv, [336](#)
 - __J_value, [336](#)
 - __N_deriv, [336](#)
 - __N_value, [336](#)
 - __Wronskian, [335](#)
 - __nu_arg, [336](#)
 - __x_arg, [336](#)
- __gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp>, [335](#)
- __gnu_cxx::__cyl_hankel_t
 - __H1_deriv, [338](#)
 - __H1_value, [338](#)
 - __H2_deriv, [338](#)
 - __H2_value, [338](#)
 - __Wronskian, [337](#)
 - __nu_arg, [338](#)
 - __x_arg, [338](#)
- __gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp>, [337](#)
- __gnu_cxx::__cyl_mod_bessel_t
 - __I_deriv, [340](#)
 - __I_value, [340](#)
 - __K_deriv, [340](#)
 - __K_value, [340](#)
 - __Wronskian, [339](#)
 - __nu_arg, [340](#)
 - __x_arg, [340](#)
- __gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp>, [339](#)
- __gnu_cxx::__fock_airy_t
 - __Wronskian, [341](#)
 - __w1_deriv, [342](#)
 - __w1_value, [342](#)
 - __w2_deriv, [342](#)
 - __w2_value, [342](#)
 - __x_arg, [342](#)
- __gnu_cxx::__fock_airy_t<_Tx, _Tp>, [341](#)
- __gnu_cxx::__fp_is_integer_t, [342](#)
 - __is_integral, [343](#)
 - __value, [343](#)
- operator bool, [343](#)
- operator(), [343](#)
- __gnu_cxx::__gamma_inc_t
 - __lgamma_value, [344](#)
 - __tgamma_value, [344](#)
- __gnu_cxx::__gamma_inc_t<_Tp>, [344](#)
- __gnu_cxx::__gamma_temme_t
 - __gamma_1_value, [346](#)
 - __gamma_2_value, [346](#)
 - __gamma_minus_value, [346](#)
 - __gamma_plus_value, [346](#)
 - __mu_arg, [346](#)
- __gnu_cxx::__gamma_temme_t<_Tp>, [345](#)
- __gnu_cxx::__jacobi_t
 - __am, [347](#)
 - __cd, [347](#)

- [__cn_value, 348](#)
- [__cs, 347](#)
- [__dc, 347](#)
- [__dn_value, 348](#)
- [__ds, 348](#)
- [__nc, 348](#)
- [__nd, 348](#)
- [__ns, 348](#)
- [__sc, 348](#)
- [__sd, 348](#)
- [__sn_value, 349](#)
- [__gnu_cxx::__jacobi_t< _Tp >, 346](#)
- [__gnu_cxx::__lgamma_t](#)
 - [__lgamma_sign, 349](#)
 - [__lgamma_value, 349](#)
- [__gnu_cxx::__lgamma_t< _Tp >, 349](#)
- [__gnu_cxx::__pqgamma_t](#)
 - [__pgamma_value, 350](#)
 - [__qgamma_value, 350](#)
- [__gnu_cxx::__pqgamma_t< _Tp >, 350](#)
- [__gnu_cxx::__quadrature_point_t](#)
 - [__quadrature_point_t, 351](#)
 - [__weight, 351](#)
 - [__zero, 351](#)
- [__gnu_cxx::__quadrature_point_t< _Tp >, 351](#)
- [__gnu_cxx::__sincos_t](#)
 - [__cos_v, 352](#)
 - [__sin_v, 352](#)
- [__gnu_cxx::__sincos_t< _Tp >, 352](#)
- [__gnu_cxx::__sph_bessel_t](#)
 - [__Wronskian, 353](#)
 - [__j_deriv, 354](#)
 - [__j_value, 354](#)
 - [__n_arg, 354](#)
 - [__n_deriv, 354](#)
 - [__n_value, 354](#)
 - [__x_arg, 354](#)
- [__gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >, 353](#)
- [__gnu_cxx::__sph_hankel_t](#)
 - [__Wronskian, 355](#)
 - [__h1_deriv, 356](#)
 - [__h1_value, 356](#)
 - [__h2_deriv, 356](#)
 - [__h2_value, 356](#)
 - [__n_arg, 356](#)
 - [__x_arg, 356](#)
- [__gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >, 355](#)
- [__gnu_cxx::__sph_mod_bessel_t](#)
 - [__Wronskian, 357](#)
 - [__i_deriv, 358](#)
 - [__i_value, 358](#)
 - [__k_deriv, 358](#)
 - [__k_value, 358](#)
 - [__x_arg, 358](#)
 - [__n_arg, 358](#)
- [__gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >, 357](#)
- [__h1_deriv](#)
 - [__gnu_cxx::__sph_hankel_t, 356](#)
- [__h1_value](#)
 - [__gnu_cxx::__sph_hankel_t, 356](#)
- [__h2_deriv](#)
 - [__gnu_cxx::__sph_hankel_t, 356](#)
- [__h2_value](#)
 - [__gnu_cxx::__sph_hankel_t, 356](#)
- [__hai_deriv](#)
 - [std::__detail::__AiryAuxilliaryState, 385](#)
- [__hai_value](#)
 - [std::__detail::__AiryAuxilliaryState, 385](#)
- [__hankel](#)
 - [std::__detail, 266](#)
- [__hankel_debye](#)
 - [std::__detail, 267](#)
- [__hankel_params](#)
 - [std::__detail, 267](#)
- [__hankel_uniform](#)
 - [std::__detail, 268](#)
- [__hankel_uniform_olver](#)
 - [std::__detail, 268](#)
- [__hankel_uniform_outer](#)
 - [std::__detail, 268](#)
- [__hankel_uniform_sum](#)
 - [std::__detail, 269](#)
- [__harmonic_number](#)
 - [std::__detail, 270](#)
- [__hermite_zeros](#)
 - [std::__detail, 270](#)
- [__heuman_lambda](#)
 - [std::__detail, 270](#)
- [__hurwitz_zeta](#)
 - [std::__detail, 270](#)
- [__hurwitz_zeta_euler_maclaurin](#)
 - [std::__detail, 271](#)
- [__hurwitz_zeta_polylog](#)
 - [std::__detail, 271](#)
- [__hydrogen](#)
 - [std::__detail, 271](#)
- [__hyperg](#)
 - [std::__detail, 272](#)
- [__hyperg_luke](#)
 - [std::__detail, 272](#)
- [__hyperg_reflect](#)
 - [std::__detail, 272](#)
- [__hyperg_series](#)
 - [std::__detail, 273](#)
- [__i_deriv](#)
 - [__gnu_cxx::__sph_mod_bessel_t, 358](#)
- [__i_value](#)
 - [__gnu_cxx::__sph_mod_bessel_t, 358](#)

- `__ibeta_cont_frac`
 - `std::__detail`, 274
- `__is_integral`
 - `__gnu_cxx::__fp_is_integer_t`, 343
- `__j_deriv`
 - `__gnu_cxx::__sph_bessel_t`, 354
- `__j_value`
 - `__gnu_cxx::__sph_bessel_t`, 354
- `__jacobi_sncndn`
 - `std::__detail`, 274
- `__jacobi_zeta`
 - `std::__detail`, 274
- `__k_deriv`
 - `__gnu_cxx::__sph_mod_bessel_t`, 358
- `__k_value`
 - `__gnu_cxx::__sph_mod_bessel_t`, 358
- `__laguerre`
 - `std::__detail`, 274
- `__laguerre_zeros`
 - `std::__detail`, 275
- `__lanczos_binet1p`
 - `std::__detail`, 275
- `__lanczos_log_gamma1p`
 - `std::__detail`, 275
- `__legendre_q`
 - `std::__detail`, 276
- `__legendre_zeros`
 - `std::__detail`, 276
- `__lgamma_sign`
 - `__gnu_cxx::__lgamma_t`, 349
- `__lgamma_value`
 - `__gnu_cxx::__gamma_inc_t`, 344
 - `__gnu_cxx::__lgamma_t`, 349
- `__log_binomial`
 - `std::__detail`, 276, 277
- `__log_binomial_sign`
 - `std::__detail`, 278
- `__log_double_factorial`
 - `std::__detail`, 278
- `__log_factorial`
 - `std::__detail`, 279
 - `std::__detail::Factorial_table`, 389
- `__log_falling_factorial`
 - `std::__detail`, 279
- `__log_gamma`
 - `std::__detail`, 279, 280
- `__log_gamma_bernoulli`
 - `std::__detail`, 280
- `__log_gamma_sign`
 - `std::__detail`, 281
- `__log_rising_factorial`
 - `std::__detail`, 281
- `__log_stirling_1`
 - `std::__detail`, 281
- `__log_stirling_1_sign`
 - `std::__detail`, 282
- `__log_stirling_2`
 - `std::__detail`, 282
- `__logint`
 - `std::__detail`, 282
- `__logistic_cdf`
 - `std::__detail`, 282
- `__logistic_pdf`
 - `std::__detail`, 283
- `__lognormal_cdf`
 - `std::__detail`, 283
- `__lognormal_pdf`
 - `std::__detail`, 283
- `__max_FGH`
 - `std::__detail`, 329
- `__max_FGH< double >`
 - `std::__detail`, 329
- `__max_FGH< float >`
 - `std::__detail`, 329
- `__mu_arg`
 - `__gnu_cxx::__gamma_temme_t`, 346
- `__n`
 - `std::__detail::Factorial_table`, 389
- `__n_arg`
 - `__gnu_cxx::__sph_bessel_t`, 354
 - `__gnu_cxx::__sph_hankel_t`, 356
- `__n_deriv`
 - `__gnu_cxx::__sph_bessel_t`, 354
- `__n_value`
 - `__gnu_cxx::__sph_bessel_t`, 354
- `__nc`
 - `__gnu_cxx::__jacobi_t`, 348
- `__nd`
 - `__gnu_cxx::__jacobi_t`, 348
- `__normal_cdf`
 - `std::__detail`, 283
- `__normal_pdf`
 - `std::__detail`, 284
- `__ns`
 - `__gnu_cxx::__jacobi_t`, 348
- `__nu_arg`
 - `__gnu_cxx::__cyl_bessel_t`, 336
 - `__gnu_cxx::__cyl_hankel_t`, 338
 - `__gnu_cxx::__cyl_mod_bessel_t`, 340
- `__owens_t`
 - `std::__detail`, 284
- `__parity`
 - `__gnu_cxx`, 181
- `__pgamma`
 - `std::__detail`, 285
- `__pgamma_value`
 - `__gnu_cxx::__pqgamma_t`, 350
- `__polar_pi`

- `std::__detail`, 285
- `__poly_hermite`
 - `std::__detail`, 285
- `__poly_hermite_asymp`
 - `std::__detail`, 286
- `__poly_hermite_recursion`
 - `std::__detail`, 286
- `__poly_jacobi`
 - `std::__detail`, 287
- `__poly_laguerre`
 - `std::__detail`, 287
- `__poly_laguerre_hyperg`
 - `std::__detail`, 288
- `__poly_laguerre_large_n`
 - `std::__detail`, 289
- `__poly_laguerre_recursion`
 - `std::__detail`, 289
- `__poly_legendre_p`
 - `std::__detail`, 290
- `__poly_prob_hermite_recursion`
 - `std::__detail`, 291
- `__poly_radial_jacobi`
 - `std::__detail`, 291
- `__polylog`
 - `std::__detail`, 292, 293
- `__polylog_exp`
 - `std::__detail`, 293
- `__polylog_exp_asymp`
 - `std::__detail`, 294
- `__polylog_exp_neg`
 - `std::__detail`, 294, 295
- `__polylog_exp_neg_int`
 - `std::__detail`, 295, 296
- `__polylog_exp_neg_real`
 - `std::__detail`, 296, 297
- `__polylog_exp_pos`
 - `std::__detail`, 297, 298
- `__polylog_exp_pos_int`
 - `std::__detail`, 299, 300
- `__polylog_exp_pos_real`
 - `std::__detail`, 300
- `__polylog_exp_sum`
 - `std::__detail`, 301
- `__psi`
 - `std::__detail`, 301, 302
- `__psi_asymp`
 - `std::__detail`, 302
- `__psi_series`
 - `std::__detail`, 303
- `__qgamma`
 - `std::__detail`, 303
- `__qgamma_value`
 - `__gnu_cxx::__pqgamma_t`, 350
- `__quadrature_point_t`
 - `__gnu_cxx::__quadrature_point_t`, 351
- `__rice_pdf`
 - `std::__detail`, 303
- `__riemann_zeta`
 - `std::__detail`, 304
- `__riemann_zeta_euler_maclaurin`
 - `std::__detail`, 304
- `__riemann_zeta_glob`
 - `std::__detail`, 304
- `__riemann_zeta_m_1`
 - `std::__detail`, 305
- `__riemann_zeta_m_1_glob`
 - `std::__detail`, 305
- `__riemann_zeta_product`
 - `std::__detail`, 305
- `__riemann_zeta_sum`
 - `std::__detail`, 306
- `__rising_factorial`
 - `std::__detail`, 306, 307
- `__sc`
 - `__gnu_cxx::__jacobi_t`, 348
- `__sd`
 - `__gnu_cxx::__jacobi_t`, 348
- `__sin_pi`
 - `std::__detail`, 307
- `__sin_v`
 - `__gnu_cxx::__sincos_t`, 352
- `__sinc`
 - `std::__detail`, 308
- `__sinc_pi`
 - `std::__detail`, 308
- `__sincos`
 - `std::__detail`, 308
- `__sincos_pi`
 - `std::__detail`, 309
- `__sincosint`
 - `std::__detail`, 309
- `__sincosint_asymp`
 - `std::__detail`, 309
- `__sincosint_cont_frac`
 - `std::__detail`, 309
- `__sincosint_series`
 - `std::__detail`, 309
- `__sinh_pi`
 - `std::__detail`, 310
- `__sinhc`
 - `std::__detail`, 310
- `__sinhc_pi`
 - `std::__detail`, 310
- `__sinhint`
 - `std::__detail`, 310
- `__sn_value`
 - `__gnu_cxx::__jacobi_t`, 349
- `__sph_bessel`

- std::__detail, 311
- __sph_bessel_ik
 - std::__detail, 312
- __sph_bessel_jn
 - std::__detail, 312
- __sph_bessel_jn_neg_arg
 - std::__detail, 313
- __sph_hankel
 - std::__detail, 313
- __sph_hankel_1
 - std::__detail, 313, 314
- __sph_hankel_2
 - std::__detail, 314, 315
- __sph_harmonic
 - std::__detail, 315
- __sph_legendre
 - std::__detail, 316
- __sph_neumann
 - std::__detail, 316, 317
- __spouge_binet1p
 - std::__detail, 317
- __spouge_log_gamma1p
 - std::__detail, 318
- __stirling_1
 - std::__detail, 319
- __stirling_1_recur
 - std::__detail, 319
- __stirling_1_series
 - std::__detail, 319
- __stirling_2
 - std::__detail, 319
- __stirling_2_recur
 - std::__detail, 320
- __stirling_2_series
 - std::__detail, 320
- __student_t_cdf
 - std::__detail, 320
- __student_t_cdfc
 - std::__detail, 321
- __student_t_pdf
 - std::__detail, 321
- __tan_pi
 - std::__detail, 321, 322
- __tanh_pi
 - std::__detail, 322
- __tgamma
 - std::__detail, 322
- __tgamma_lower
 - std::__detail, 323
- __tgamma_value
 - __gnu_cxx::__gamma_inc_t, 344
- __theta_1
 - std::__detail, 323
- __theta_2
 - std::__detail, 323
- __theta_2_asymp
 - std::__detail, 324
- __theta_2_sum
 - std::__detail, 324
- __theta_3
 - std::__detail, 324
- __theta_3_asymp
 - std::__detail, 325
- __theta_3_sum
 - std::__detail, 325
- __theta_4
 - std::__detail, 325
- __theta_c
 - std::__detail, 326
- __theta_d
 - std::__detail, 326
- __theta_n
 - std::__detail, 326
- __theta_s
 - std::__detail, 326
- __tricoli_u
 - std::__detail, 326
- __tricoli_u_naive
 - std::__detail, 327
- __value
 - __gnu_cxx::__fp_is_integer_t, 343
- __w1_deriv
 - __gnu_cxx::__fock_airy_t, 342
- __w1_value
 - __gnu_cxx::__fock_airy_t, 342
- __w2_deriv
 - __gnu_cxx::__fock_airy_t, 342
- __w2_value
 - __gnu_cxx::__fock_airy_t, 342
- __weibull_cdf
 - std::__detail, 327
- __weibull_pdf
 - std::__detail, 328
- __weight
 - __gnu_cxx::__quadrature_point_t, 351
- __x_arg
 - __gnu_cxx::__airy_t, 334
 - __gnu_cxx::__cyl_bessel_t, 336
 - __gnu_cxx::__cyl_hankel_t, 338
 - __gnu_cxx::__cyl_mod_bessel_t, 340
 - __gnu_cxx::__fock_airy_t, 342
 - __gnu_cxx::__sph_bessel_t, 354
 - __gnu_cxx::__sph_hankel_t, 356
 - __gnu_cxx::__sph_mod_bessel_t, 358
- __z
 - std::__detail::__AiryAuxilliaryState, 385
 - std::__detail::__AiryState, 387
- __zernike

- std::__detail, [328](#)
- __zero
 - __gnu_cxx::__quadrature_point_t, [351](#)
- __znorm1
 - std::__detail, [329](#)
- __znorm2
 - std::__detail, [329](#)
- airy_ai
 - GNU Extended Mathematical Special Functions, [57](#), [58](#)
- airy_aif
 - GNU Extended Mathematical Special Functions, [58](#)
- airy_ail
 - GNU Extended Mathematical Special Functions, [58](#)
- airy_bi
 - GNU Extended Mathematical Special Functions, [58](#), [59](#)
- airy_bif
 - GNU Extended Mathematical Special Functions, [59](#)
- airy_bil
 - GNU Extended Mathematical Special Functions, [60](#)
- 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, [60](#)
- bernoullif
 - GNU Extended Mathematical Special Functions, [61](#)
- bernoullil
 - GNU Extended Mathematical Special Functions, [61](#)
- beta
 - C++17/IS29124 Mathematical Special Functions, [24](#)
- betaf
 - C++17/IS29124 Mathematical Special Functions, [25](#)
- betal
 - C++17/IS29124 Mathematical Special Functions, [25](#)
- binomial
 - GNU Extended Mathematical Special Functions, [61](#)
- binomial_cdf
 - GNU Extended Mathematical Special Functions, [62](#)
- binomial_pdf
 - GNU Extended Mathematical Special Functions, [62](#)
- binomialf
 - GNU Extended Mathematical Special Functions, [63](#)
- binomiall
 - GNU Extended Mathematical Special Functions, [63](#)
- bits/sf_airy.tcc, [391](#)
- bits/sf_bernoulli.tcc, [393](#)
- bits/sf_bessel.tcc, [394](#)
- bits/sf_beta.tcc, [397](#)
- bits/sf_cardinal.tcc, [399](#)
- bits/sf_chebyshev.tcc, [400](#)
- bits/sf_dawson.tcc, [402](#)
- bits/sf_distributions.tcc, [403](#)
- bits/sf_ellint.tcc, [406](#)
- bits/sf_euler.tcc, [409](#)
- bits/sf_expint.tcc, [410](#)
- bits/sf_fresnel.tcc, [413](#)
- bits/sf_gamma.tcc, [414](#)
- bits/sf_gegenbauer.tcc, [422](#)
- bits/sf_hankel.tcc, [423](#)
- bits/sf_hermite.tcc, [426](#)
- bits/sf_hydrogen.tcc, [428](#)
- bits/sf_hyperg.tcc, [429](#)
- bits/sf_hypint.tcc, [431](#)
- bits/sf_jacobi.tcc, [433](#)
- bits/sf_laguerre.tcc, [435](#)
- bits/sf_legendre.tcc, [436](#)
- bits/sf_mod_bessel.tcc, [438](#)
- bits/sf_owens_t.tcc, [440](#)
- bits/sf_polylog.tcc, [441](#)
- bits/sf_stirling.tcc, [444](#)
- bits/sf_theta.tcc, [446](#)
- bits/sf_trig.tcc, [448](#)
- bits/sf_trigint.tcc, [450](#)
- bits/sf_zeta.tcc, [451](#)
- bits/specfun.h, [454](#)
- bits/specfun_state.h, [469](#)
- bose_einstein
 - GNU Extended Mathematical Special Functions, [63](#)
- bose_einsteinf
 - GNU Extended Mathematical Special Functions, [63](#)
- bose_einsteinl
 - GNU Extended Mathematical Special Functions, [63](#)
- 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, [39](#)
- laguerrel, [39](#)
- legendre, [39](#)
- legendref, [40](#)
- legendrel, [40](#)
- riemann_zeta, [40](#)
- riemann_zetaf, [41](#)
- riemann_zetal, [41](#)
- sph_bessel, [41](#)
- sph_besself, [42](#)
- sph_bessell, [42](#)
- sph_legendre, [42](#)
- sph_legendref, [43](#)
- sph_legendrel, [43](#)
- sph_neumann, [43](#)
- sph_neumannf, [44](#)
- sph_neumannl, [44](#)
- chebyshev_t
 - GNU Extended Mathematical Special Functions, [63](#)
- chebyshev_tf
 - GNU Extended Mathematical Special Functions, [64](#)
- chebyshev_tl
 - GNU Extended Mathematical Special Functions, [64](#)
- chebyshev_u
 - GNU Extended Mathematical Special Functions, [64](#)
- chebyshev_uf
 - GNU Extended Mathematical Special Functions, [65](#)
- chebyshev_ul
 - GNU Extended Mathematical Special Functions, [65](#)
- chebyshev_v
 - GNU Extended Mathematical Special Functions, [65](#)
- chebyshev_vf
 - GNU Extended Mathematical Special Functions, [66](#)
- chebyshev_vl
 - GNU Extended Mathematical Special Functions, [66](#)
- chebyshev_w
 - GNU Extended Mathematical Special Functions, [66](#)
- chebyshev_wf
 - GNU Extended Mathematical Special Functions, [67](#)
- chebyshev_wl
 - GNU Extended Mathematical Special Functions, [67](#)
- clausen
 - GNU Extended Mathematical Special Functions, [67](#), [68](#)
- clausen_cl
 - GNU Extended Mathematical Special Functions, [68](#)
- clausen_clf
 - GNU Extended Mathematical Special Functions, [69](#)
- clausen_cll
 - GNU Extended Mathematical Special Functions, [69](#)
- clausen_sl
 - GNU Extended Mathematical Special Functions, [69](#)
- clausen_slf
 - GNU Extended Mathematical Special Functions, [70](#)
- clausen_sll
 - GNU Extended Mathematical Special Functions, [70](#)
- clausenf
 - GNU Extended Mathematical Special Functions, [70](#)
- clausenl
 - GNU Extended Mathematical Special Functions, [71](#)
- 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](#)

- 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, [71](#)
- `comp_ellint_df`
 - GNU Extended Mathematical Special Functions, [72](#)
- `comp_ellint_dl`
 - GNU Extended Mathematical Special Functions, [72](#)
- `comp_ellint_rf`
 - GNU Extended Mathematical Special Functions, [72](#)
- `comp_ellint_rg`
 - GNU Extended Mathematical Special Functions, [73](#)
- `conf_hyperg`
 - GNU Extended Mathematical Special Functions, [74](#)
- `conf_hyperg_lim`
 - GNU Extended Mathematical Special Functions, [74](#)
- `conf_hyperg_limf`
 - GNU Extended Mathematical Special Functions, [74](#)
- `conf_hyperg_liml`
 - GNU Extended Mathematical Special Functions, [75](#)
- `conf_hypergf`
 - GNU Extended Mathematical Special Functions, [75](#)
- `conf_hypergl`
 - GNU Extended Mathematical Special Functions, [75](#)
- `cos_pi`
 - GNU Extended Mathematical Special Functions, [75](#)
- `cos_pif`
 - GNU Extended Mathematical Special Functions, [76](#)
- `cos_pil`
 - GNU Extended Mathematical Special Functions, [76](#)
- `cosh_pi`
 - GNU Extended Mathematical Special Functions, [76](#)
- `cosh_pif`
 - GNU Extended Mathematical Special Functions, [77](#)
- `cosh_pil`
 - GNU Extended Mathematical Special Functions, [77](#)
- `coshint`
 - GNU Extended Mathematical Special Functions, [77](#)
- `coshintf`
 - GNU Extended Mathematical Special Functions, [78](#)
- `coshintl`
 - GNU Extended Mathematical Special Functions, [78](#)
- `cosint`
 - GNU Extended Mathematical Special Functions, [78](#)
- `cosintf`
 - GNU Extended Mathematical Special Functions, [78](#)
- `cosintl`
 - GNU Extended Mathematical Special Functions, [79](#)
- `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, [79](#), [80](#)
- `cyl_hankel_1f`
 - GNU Extended Mathematical Special Functions, [80](#)
- `cyl_hankel_1l`
 - GNU Extended Mathematical Special Functions, [80](#), [81](#)
- `cyl_hankel_2`
 - GNU Extended Mathematical Special Functions, [81](#), [82](#)
- `cyl_hankel_2f`
 - GNU Extended Mathematical Special Functions, [82](#)
- `cyl_hankel_2l`
 - GNU Extended Mathematical Special Functions, [82](#), [83](#)
- `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, [83](#)
- `dawsonf`
 - GNU Extended Mathematical Special Functions, [83](#)
- `dawsonl`
 - GNU Extended Mathematical Special Functions, [84](#)
- `debye`
 - GNU Extended Mathematical Special Functions, [84](#)
- `debyef`
 - GNU Extended Mathematical Special Functions, [84](#)
- `debyel`
 - GNU Extended Mathematical Special Functions, [84](#)
- `dilog`
 - GNU Extended Mathematical Special Functions, [85](#)
- `dilogf`
 - GNU Extended Mathematical Special Functions, [85](#)

- dilogl
 - GNU Extended Mathematical Special Functions, [85](#)
- dirichlet_beta
 - GNU Extended Mathematical Special Functions, [85](#)
- dirichlet_betaf
 - GNU Extended Mathematical Special Functions, [86](#)
- dirichlet_betal
 - GNU Extended Mathematical Special Functions, [86](#)
- dirichlet_eta
 - GNU Extended Mathematical Special Functions, [86](#)
- dirichlet_etaf
 - GNU Extended Mathematical Special Functions, [87](#)
- dirichlet_etald
 - GNU Extended Mathematical Special Functions, [87](#)
- dirichlet_lambda
 - GNU Extended Mathematical Special Functions, [87](#)
- dirichlet_lambdaf
 - GNU Extended Mathematical Special Functions, [88](#)
- dirichlet_lambdal
 - GNU Extended Mathematical Special Functions, [88](#)
- double_factorial
 - GNU Extended Mathematical Special Functions, [88](#)
- double_factorialf
 - GNU Extended Mathematical Special Functions, [89](#)
- double_factoriall
 - GNU Extended Mathematical Special Functions, [89](#)
- 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, [89](#)
- ellint_celf
 - GNU Extended Mathematical Special Functions, [90](#)
- ellint_cell
 - GNU Extended Mathematical Special Functions, [90](#)
- ellint_d
 - GNU Extended Mathematical Special Functions, [90](#)
- ellint_df
 - GNU Extended Mathematical Special Functions, [91](#)
- ellint_dl
 - GNU Extended Mathematical Special Functions, [91](#)
- ellint_el1
 - GNU Extended Mathematical Special Functions, [91](#)
- ellint_el1f
 - GNU Extended Mathematical Special Functions, [92](#)
- ellint_el1l
 - GNU Extended Mathematical Special Functions, [92](#)
- ellint_el2
 - GNU Extended Mathematical Special Functions, [92](#)
- ellint_el2f
 - GNU Extended Mathematical Special Functions, [92](#)
- ellint_el2l
 - GNU Extended Mathematical Special Functions, [93](#)
- ellint_el3
 - GNU Extended Mathematical Special Functions, [93](#)
- ellint_el3f
 - GNU Extended Mathematical Special Functions, [93](#)
- ellint_el3l
 - GNU Extended Mathematical Special Functions, [94](#)
- ellint_rc
 - GNU Extended Mathematical Special Functions, [94](#)
- ellint_rcf
 - GNU Extended Mathematical Special Functions, [95](#)
- ellint_rcl
 - GNU Extended Mathematical Special Functions, [95](#)
- ellint_rd
 - GNU Extended Mathematical Special Functions, [95](#)
- ellint_rdf
 - GNU Extended Mathematical Special Functions, [96](#)
- ellint_rdl
 - GNU Extended Mathematical Special Functions, [96](#)
- ellint_rf
 - GNU Extended Mathematical Special Functions, [96](#)
- ellint_rff
 - GNU Extended Mathematical Special Functions, [96](#)
- ellint_rfl
 - GNU Extended Mathematical Special Functions, [97](#)
- ellint_rg
 - GNU Extended Mathematical Special Functions, [97](#)
- ellint_rgf
 - GNU Extended Mathematical Special Functions, [98](#)
- ellint_rgl
 - GNU Extended Mathematical Special Functions, [98](#)
- ellint_rj
 - GNU Extended Mathematical Special Functions, [98](#)
- ellint_rjf
 - GNU Extended Mathematical Special Functions, [99](#)
- ellint_rjl
 - GNU Extended Mathematical Special Functions, [99](#)
- ellnome
 - GNU Extended Mathematical Special Functions, [99](#)
- ellnomef
 - GNU Extended Mathematical Special Functions, [99](#)

- GNU Extended Mathematical Special Functions, [99](#)
- ellnomel
 - GNU Extended Mathematical Special Functions, [100](#)
- euler
 - GNU Extended Mathematical Special Functions, [100](#)
- eulerian_1
 - GNU Extended Mathematical Special Functions, [100](#)
- eulerian_2
 - GNU Extended Mathematical Special Functions, [100](#)
- expint
 - C++17/IS29124 Mathematical Special Functions, [36](#)
 - GNU Extended Mathematical Special Functions, [101](#)
- expintf
 - C++17/IS29124 Mathematical Special Functions, [36](#)
 - GNU Extended Mathematical Special Functions, [101](#)
- expintl
 - C++17/IS29124 Mathematical Special Functions, [36](#)
 - GNU Extended Mathematical Special Functions, [101](#)
- exponential_cdf
 - GNU Extended Mathematical Special Functions, [102](#)
- exponential_pdf
 - GNU Extended Mathematical Special Functions, [102](#)
- ext/math_util.h, [471](#)
- factorial
 - GNU Extended Mathematical Special Functions, [102](#)
- factorialf
 - GNU Extended Mathematical Special Functions, [102](#)
- factoriall
 - GNU Extended Mathematical Special Functions, [103](#)
- falling_factorial
 - GNU Extended Mathematical Special Functions, [103](#)
- falling_factorialf
 - GNU Extended Mathematical Special Functions, [103](#)
- falling_factoriall
 - GNU Extended Mathematical Special Functions, [103](#)
- fermi_dirac
 - GNU Extended Mathematical Special Functions, [104](#)
- fermi_diracf
 - GNU Extended Mathematical Special Functions, [104](#)
- fermi_diracl
 - GNU Extended Mathematical Special Functions, [104](#)
- fisher_f_cdf
 - GNU Extended Mathematical Special Functions, [104](#)
- fisher_f_pdf
 - GNU Extended Mathematical Special Functions, [104](#)
- fresnel_c
 - GNU Extended Mathematical Special Functions, [105](#)
- fresnel_cf
 - GNU Extended Mathematical Special Functions, [105](#)
- fresnel_cl
 - GNU Extended Mathematical Special Functions, [105](#)
- fresnel_s
 - GNU Extended Mathematical Special Functions, [105](#)
- fresnel_sf
 - GNU Extended Mathematical Special Functions, [106](#)
- fresnel_sl
 - GNU Extended Mathematical Special Functions, [106](#)
- GNU Extended Mathematical Special Functions, [45](#)
 - airy_ai, [57](#), [58](#)
 - airy_aif, [58](#)
 - airy_ail, [58](#)
 - airy_bi, [58](#), [59](#)
 - airy_bif, [59](#)
 - airy_bil, [60](#)
 - bernoulli, [60](#)
 - bernoullif, [61](#)
 - bernoullil, [61](#)
 - binomial, [61](#)
 - binomial_cdf, [62](#)
 - binomial_pdf, [62](#)
 - binomialf, [63](#)
 - binomiall, [63](#)
 - bose_einstein, [63](#)
 - bose_einsteinf, [63](#)
 - bose_einsteinl, [63](#)
 - chebyshev_t, [63](#)
 - chebyshev_tf, [64](#)
 - chebyshev_tl, [64](#)
 - chebyshev_u, [64](#)
 - chebyshev_uf, [65](#)
 - chebyshev_ul, [65](#)
 - chebyshev_v, [65](#)
 - chebyshev_vf, [66](#)
 - chebyshev_vl, [66](#)
 - chebyshev_w, [66](#)
 - chebyshev_wf, [67](#)
 - chebyshev_wl, [67](#)
 - clausen, [67](#), [68](#)
 - clausen_cl, [68](#)
 - clausen_clf, [69](#)
 - clausen_cll, [69](#)
 - clausen_sl, [69](#)
 - clausen_slf, [70](#)
 - clausen_sll, [70](#)
 - clausenf, [70](#)
 - clausenl, [71](#)
 - comp_ellint_d, [71](#)
 - comp_ellint_df, [72](#)
 - comp_ellint_dl, [72](#)
 - comp_ellint_rf, [72](#)
 - comp_ellint_rg, [73](#)
 - conf_hyperg, [74](#)
 - conf_hyperg_lim, [74](#)
 - conf_hyperg_limf, [74](#)
 - conf_hyperg_liml, [75](#)
 - conf_hypergf, [75](#)

conf_hypergl, 75
 cos_pi, 75
 cos_pif, 76
 cos_pil, 76
 cosh_pi, 76
 cosh_pif, 77
 cosh_pil, 77
 coshint, 77
 coshintf, 78
 coshintl, 78
 cosint, 78
 cosintf, 78
 cosintl, 79
 cyl_hankel_1, 79, 80
 cyl_hankel_1f, 80
 cyl_hankel_1l, 80, 81
 cyl_hankel_2, 81, 82
 cyl_hankel_2f, 82
 cyl_hankel_2l, 82, 83
 dawson, 83
 dawsonf, 83
 dawsonl, 84
 debye, 84
 debyef, 84
 debyel, 84
 dilog, 85
 dilogf, 85
 dilogl, 85
 dirichlet_beta, 85
 dirichlet_betaf, 86
 dirichlet_betall, 86
 dirichlet_eta, 86
 dirichlet_etaf, 87
 dirichlet_etall, 87
 dirichlet_lambda, 87
 dirichlet_lambdaf, 88
 dirichlet_lambdal, 88
 double_factorial, 88
 double_factorialf, 89
 double_factoriall, 89
 ellint_cel, 89
 ellint_celf, 90
 ellint_cell, 90
 ellint_d, 90
 ellint_df, 91
 ellint_dl, 91
 ellint_el1, 91
 ellint_el1f, 92
 ellint_el1l, 92
 ellint_el2, 92
 ellint_el2f, 92
 ellint_el2l, 93
 ellint_el3, 93
 ellint_el3f, 93
 ellint_el3l, 94
 ellint_rc, 94
 ellint_rcf, 95
 ellint_rcl, 95
 ellint_rd, 95
 ellint_rdf, 96
 ellint_rdl, 96
 ellint_rf, 96
 ellint_rff, 96
 ellint_rfl, 97
 ellint_rg, 97
 ellint_rgf, 98
 ellint_rgl, 98
 ellint_rj, 98
 ellint_rjf, 99
 ellint_rjl, 99
 ellnome, 99
 ellnomef, 99
 ellnomel, 100
 euler, 100
 eulerian_1, 100
 eulerian_2, 100
 expint, 101
 expintf, 101
 expintl, 101
 exponential_cdf, 102
 exponential_pdf, 102
 factorial, 102
 factorialf, 102
 factoriall, 103
 falling_factorial, 103
 falling_factorialf, 103
 falling_factoriall, 103
 fermi_dirac, 104
 fermi_diracf, 104
 fermi_diracl, 104
 fisher_f_cdf, 104
 fisher_f_pdf, 104
 fresnel_c, 105
 fresnel_cf, 105
 fresnel_cl, 105
 fresnel_s, 105
 fresnel_sf, 106
 fresnel_sl, 106
 gamma_cdf, 106
 gamma_pdf, 106
 gamma_reciprocal, 106
 gamma_reciprocalf, 107
 gamma_reciprocall, 107
 gegenbauer, 107
 gegenbauerf, 108
 gegenbauerl, 108
 harmonic, 108
 heuman_lambda, 109

heuman_lambdaf, 109
heuman_lambdal, 109
hurwitz_zeta, 109, 111
hurwitz_zetaf, 111
hurwitz_zetal, 111
hyperg, 111
hypergf, 112
hypergl, 112
ibeta, 112
ibetac, 113
ibetacf, 113
ibetacl, 113
ibetaf, 113
ibetal, 114
jacobi, 114
jacobi_cn, 114
jacobi_cnf, 115
jacobi_cnl, 115
jacobi_dn, 115
jacobi_dnf, 116
jacobi_dnl, 116
jacobi_sn, 116
jacobi_snf, 117
jacobi_snl, 117
jacobi_zeta, 117
jacobi_zetaf, 118
jacobi_zetal, 118
jacobif, 118
jacobil, 118
lbinomial, 119
lbinomialf, 119
lbinomiall, 119
ldouble_factorial, 120
ldouble_factorialf, 120
ldouble_factoriall, 120
legendre_q, 120
legendre_qf, 121
legendre_ql, 121
lfactorial, 121
lfactorialf, 121
lfactoriall, 122
lfalling_factorial, 122
lfalling_factorialf, 122
lfalling_factoriall, 122
lgamma, 123
lgammaf, 123
lgammal, 123, 124
logint, 124
logintf, 124
logintl, 124
logistic_cdf, 125
logistic_pdf, 125
lognormal_cdf, 125
lognormal_pdf, 125
lrising_factorial, 126
lrising_factorialf, 126
lrising_factoriall, 126
normal_cdf, 126
normal_pdf, 127
owens_t, 127
owens_tf, 127
owens_tl, 128
pgamma, 128
pgammaf, 128
pgammal, 128
polylog, 128, 129
polylogf, 129
polylogl, 129
psi, 130
psif, 130
psil, 130
qgamma, 130
qgammaf, 131
qgammal, 131
radpoly, 131
radpolyf, 131
radpolyl, 132
rising_factorial, 132
rising_factorialf, 132
rising_factoriall, 132
sin_pi, 133
sin_pif, 133
sin_pil, 133
sinc, 133
sinc_pi, 134
sinc_pif, 134
sinc_pil, 134
sincf, 135
sincl, 135
sincos, 135
sincos_pi, 136
sincos_pif, 136
sincos_pil, 136
sincosf, 136
sincosl, 136
sinh_pi, 137
sinh_pif, 137
sinh_pil, 137
sinhc, 137
sinhc_pi, 138
sinhc_pif, 138
sinhc_pil, 138
sinhcf, 139
sinhcl, 139
sinhint, 139
sinhintf, 140
sinhintl, 140
sinint, 140

- sinintf, [140](#)
- sinintl, [141](#)
- sph_bessel_i, [141](#)
- sph_bessel_if, [142](#)
- sph_bessel_il, [142](#)
- sph_bessel_k, [142](#)
- sph_bessel_kf, [143](#)
- sph_bessel_kl, [143](#)
- sph_hankel_1, [143](#), [144](#)
- sph_hankel_1f, [144](#)
- sph_hankel_1l, [144](#), [145](#)
- sph_hankel_2, [145](#), [146](#)
- sph_hankel_2f, [146](#)
- sph_hankel_2l, [146](#), [147](#)
- sph_harmonic, [147](#)
- sph_harmonicf, [147](#)
- sph_harmonicl, [148](#)
- stirling_1, [148](#)
- stirling_2, [148](#)
- student_t_cdf, [149](#)
- student_t_pdf, [149](#)
- tan_pi, [150](#)
- tan_pif, [150](#)
- tan_pil, [150](#)
- tanh_pi, [150](#)
- tanh_pif, [151](#)
- tanh_pil, [151](#)
- tgamma, [151](#)
- tgamma_lower, [152](#)
- tgamma_lowerf, [152](#)
- tgamma_lowerl, [152](#)
- tgammaf, [152](#), [153](#)
- tgammal, [153](#), [154](#)
- theta_1, [154](#)
- theta_1f, [154](#)
- theta_1l, [154](#)
- theta_2, [155](#)
- theta_2f, [155](#)
- theta_2l, [155](#)
- theta_3, [155](#)
- theta_3f, [156](#)
- theta_3l, [156](#)
- theta_4, [156](#)
- theta_4f, [157](#)
- theta_4l, [157](#)
- theta_c, [157](#)
- theta_cf, [158](#)
- theta_cl, [158](#)
- theta_d, [158](#)
- theta_df, [159](#)
- theta_dl, [159](#)
- theta_n, [159](#)
- theta_nf, [160](#)
- theta_nl, [160](#)
- theta_s, [160](#)
- theta_sf, [161](#)
- theta_sl, [161](#)
- tricomi_u, [161](#)
- tricomi_uf, [162](#)
- tricomi_ul, [162](#)
- weibull_cdf, [162](#)
- weibull_pdf, [162](#)
- zernike, [163](#)
- zernikef, [163](#)
- zernikel, [164](#)
- gamma_cdf
 - GNU Extended Mathematical Special Functions, [106](#)
- gamma_pdf
 - GNU Extended Mathematical Special Functions, [106](#)
- gamma_reciprocal
 - GNU Extended Mathematical Special Functions, [106](#)
- gamma_reciprocalf
 - GNU Extended Mathematical Special Functions, [107](#)
- gamma_reciprocall
 - GNU Extended Mathematical Special Functions, [107](#)
- gegenbauer
 - GNU Extended Mathematical Special Functions, [107](#)
- gegenbauerf
 - GNU Extended Mathematical Special Functions, [108](#)
- gegenbauerl
 - GNU Extended Mathematical Special Functions, [108](#)
- harmonic
 - GNU Extended Mathematical Special Functions, [108](#)
- 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, [109](#)
- heuman_lambdaf
 - GNU Extended Mathematical Special Functions, [109](#)
- heuman_lambdal
 - GNU Extended Mathematical Special Functions, [109](#)
- hurwitz_zeta
 - GNU Extended Mathematical Special Functions, [109](#), [111](#)
- hurwitz_zetaf
 - GNU Extended Mathematical Special Functions, [111](#)
- hurwitz_zetal
 - GNU Extended Mathematical Special Functions, [111](#)
- hyperg
 - GNU Extended Mathematical Special Functions, [111](#)
- hypergf
 - GNU Extended Mathematical Special Functions, [112](#)
- hypergl

- GNU Extended Mathematical Special Functions, [112](#)
- ibeta
 - GNU Extended Mathematical Special Functions, [112](#)
- ibetac
 - GNU Extended Mathematical Special Functions, [113](#)
- ibetacf
 - GNU Extended Mathematical Special Functions, [113](#)
- ibetacl
 - GNU Extended Mathematical Special Functions, [113](#)
- ibetaf
 - GNU Extended Mathematical Special Functions, [113](#)
- ibetal
 - GNU Extended Mathematical Special Functions, [114](#)
- inner_radius
 - std::__detail::__Airy, [366](#)
 - std::__detail::__Airy_default_radii< double >, [376](#)
 - std::__detail::__Airy_default_radii< float >, [376](#)
 - std::__detail::__Airy_default_radii< long double >, [377](#)
- jacobi
 - GNU Extended Mathematical Special Functions, [114](#)
- jacobi_cn
 - GNU Extended Mathematical Special Functions, [114](#)
- jacobi_cnf
 - GNU Extended Mathematical Special Functions, [115](#)
- jacobi_cnl
 - GNU Extended Mathematical Special Functions, [115](#)
- jacobi_dn
 - GNU Extended Mathematical Special Functions, [115](#)
- jacobi_dnf
 - GNU Extended Mathematical Special Functions, [116](#)
- jacobi_dnl
 - GNU Extended Mathematical Special Functions, [116](#)
- jacobi_sn
 - GNU Extended Mathematical Special Functions, [116](#)
- jacobi_snf
 - GNU Extended Mathematical Special Functions, [117](#)
- jacobi_snl
 - GNU Extended Mathematical Special Functions, [117](#)
- jacobi_zeta
 - GNU Extended Mathematical Special Functions, [117](#)
- jacobi_zetaf
 - GNU Extended Mathematical Special Functions, [118](#)
- jacobi_zetal
 - GNU Extended Mathematical Special Functions, [118](#)
- jacobiif
 - GNU Extended Mathematical Special Functions, [118](#)
- jacobil
 - GNU Extended Mathematical Special Functions, [118](#)
- laguerre
 - C++17/IS29124 Mathematical Special Functions, [37](#)
- laguerref
 - C++17/IS29124 Mathematical Special Functions, [39](#)
- laguerrel
 - C++17/IS29124 Mathematical Special Functions, [39](#)
- lbinomial
 - GNU Extended Mathematical Special Functions, [119](#)
- lbinomialf
 - GNU Extended Mathematical Special Functions, [119](#)
- lbinomiall
 - GNU Extended Mathematical Special Functions, [119](#)
- ldouble_factorial
 - GNU Extended Mathematical Special Functions, [120](#)
- ldouble_factorialf
 - GNU Extended Mathematical Special Functions, [120](#)
- ldouble_factoriall
 - GNU Extended Mathematical Special Functions, [120](#)
- legendre
 - C++17/IS29124 Mathematical Special Functions, [39](#)
- legendre_q
 - GNU Extended Mathematical Special Functions, [120](#)
- legendre_qf
 - GNU Extended Mathematical Special Functions, [121](#)
- legendre_ql
 - GNU Extended Mathematical Special Functions, [121](#)
- legendref
 - C++17/IS29124 Mathematical Special Functions, [40](#)
- legendrel
 - C++17/IS29124 Mathematical Special Functions, [40](#)
- lfactorial
 - GNU Extended Mathematical Special Functions, [121](#)
- lfactorialf
 - GNU Extended Mathematical Special Functions, [121](#)
- lfactoriall
 - GNU Extended Mathematical Special Functions, [122](#)
- lfalling_factorial
 - GNU Extended Mathematical Special Functions, [122](#)
- lfalling_factorialf
 - GNU Extended Mathematical Special Functions, [122](#)
- lfalling_factoriall
 - GNU Extended Mathematical Special Functions, [122](#)
- lgamma
 - GNU Extended Mathematical Special Functions, [123](#)
- lgammaf
 - GNU Extended Mathematical Special Functions, [123](#)
- lgammal
 - GNU Extended Mathematical Special Functions, [123](#), [124](#)
- logint
 - GNU Extended Mathematical Special Functions, [124](#)
- logintf
 - GNU Extended Mathematical Special Functions, [124](#)
- logintl
 - GNU Extended Mathematical Special Functions, [124](#)
- logistic_cdf
 - GNU Extended Mathematical Special Functions, [125](#)

- logistic_pdf
 - GNU Extended Mathematical Special Functions, [125](#)
- lognormal_cdf
 - GNU Extended Mathematical Special Functions, [125](#)
- lognormal_pdf
 - GNU Extended Mathematical Special Functions, [125](#)
- lrising_factorial
 - GNU Extended Mathematical Special Functions, [126](#)
- lrising_factorialf
 - GNU Extended Mathematical Special Functions, [126](#)
- lrising_factoriall
 - GNU Extended Mathematical Special Functions, [126](#)
- n_arg
 - __gnu_cxx::__sph_mod_bessel_t, [358](#)
- normal_cdf
 - GNU Extended Mathematical Special Functions, [126](#)
- normal_pdf
 - GNU Extended Mathematical Special Functions, [127](#)
- operator bool
 - __gnu_cxx::__fp_is_integer_t, [343](#)
- operator()
 - __gnu_cxx::__fp_is_integer_t, [343](#)
 - std::__detail::Airy, [365](#)
 - std::__detail::Airy_asymp, [369](#)
 - std::__detail::Airy_asymp_series, [374](#)
 - std::__detail::AsympTerminator, [388](#)
 - std::__detail::Terminator, [390](#)
- outer_radius
 - std::__detail::Airy, [366](#)
 - std::__detail::Airy_default_radii< double >, [376](#)
 - std::__detail::Airy_default_radii< float >, [376](#)
 - std::__detail::Airy_default_radii< long double >, [377](#)
- owens_t
 - GNU Extended Mathematical Special Functions, [127](#)
- owens_tf
 - GNU Extended Mathematical Special Functions, [127](#)
- owens_tl
 - GNU Extended Mathematical Special Functions, [128](#)
- pgamma
 - GNU Extended Mathematical Special Functions, [128](#)
- pgammaf
 - GNU Extended Mathematical Special Functions, [128](#)
- pgammal
 - GNU Extended Mathematical Special Functions, [128](#)
- polylog
 - GNU Extended Mathematical Special Functions, [128](#), [129](#)
- polylogf
 - GNU Extended Mathematical Special Functions, [129](#)
- polylogl
 - GNU Extended Mathematical Special Functions, [129](#)
- psi
 - GNU Extended Mathematical Special Functions, [130](#)
- psif
 - GNU Extended Mathematical Special Functions, [130](#)
- psil
 - GNU Extended Mathematical Special Functions, [130](#)
- qgamma
 - GNU Extended Mathematical Special Functions, [130](#)
- qgammaf
 - GNU Extended Mathematical Special Functions, [131](#)
- qgammal
 - GNU Extended Mathematical Special Functions, [131](#)
- radpoly
 - GNU Extended Mathematical Special Functions, [131](#)
- radpolyf
 - GNU Extended Mathematical Special Functions, [131](#)
- radpolyl
 - GNU Extended Mathematical Special Functions, [132](#)
- riemann_zeta
 - C++17/IS29124 Mathematical Special Functions, [40](#)
- riemann_zetaf
 - C++17/IS29124 Mathematical Special Functions, [41](#)
- riemann_zetal
 - C++17/IS29124 Mathematical Special Functions, [41](#)
- rising_factorial
 - GNU Extended Mathematical Special Functions, [132](#)
- rising_factorialf
 - GNU Extended Mathematical Special Functions, [132](#)
- rising_factoriall
 - GNU Extended Mathematical Special Functions, [132](#)
- scalar_type
 - std::__detail::Airy, [365](#)
 - std::__detail::Airy_asymp_series, [374](#)
- sf_airy.tcc
 - _GLIBCXX_BITS_SF_AIRY_TCC, [393](#)
- sf_bernoulli.tcc
 - _GLIBCXX_BITS_SF_BERNOULLI_TCC, [394](#)
- sf_bessel.tcc
 - _GLIBCXX_BITS_SF_BESSEL_TCC, [397](#)
- sf_beta.tcc
 - _GLIBCXX_BITS_SF_BETA_TCC, [398](#)
- sf_cardinal.tcc
 - _GLIBCXX_BITS_SF_CARDINAL_TCC, [400](#)
- sf_chebyshev.tcc
 - _GLIBCXX_BITS_SF_CHEBYSHEV_TCC, [402](#)
- sf_dawson.tcc
 - _GLIBCXX_BITS_SF_DAWSON_TCC, [403](#)
- sf_distributions.tcc
 - _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC, [406](#)
- sf_ellint.tcc
 - _GLIBCXX_BITS_SF_ELLINT_TCC, [408](#)
- sf_euler.tcc

- `_GLIBCXX_BITS_SF_EULER_TCC`, 410
- `sf_expint.tcc`
 - `_GLIBCXX_BITS_SF_EXPINT_TCC`, 412
- `sf_fresnel.tcc`
 - `_GLIBCXX_BITS_SF_FRESNEL_TCC`, 414
- `sf_gamma.tcc`
 - `_GLIBCXX_BITS_SF_GAMMA_TCC`, 422
- `sf_gegenbauer.tcc`
 - `_GLIBCXX_BITS_SF_GEGENBAUER_TCC`, 423
- `sf_hankel.tcc`
 - `_GLIBCXX_BITS_SF_HANKEL_TCC`, 426
- `sf_hermite.tcc`
 - `_GLIBCXX_BITS_SF_HERMITE_TCC`, 428
- `sf_hydrogen.tcc`
 - `_GLIBCXX_BITS_SF_HYDROGEN_TCC`, 429
- `sf_hyperg.tcc`
 - `_GLIBCXX_BITS_SF_HYPERG_TCC`, 431
- `sf_hypint.tcc`
 - `_GLIBCXX_BITS_SF_HYPINT_TCC`, 433
- `sf_jacobi.tcc`
 - `_GLIBCXX_BITS_SF_JACOBI_TCC`, 434
- `sf_laguerre.tcc`
 - `_GLIBCXX_BITS_SF_LAGUERRE_TCC`, 436
- `sf_legendre.tcc`
 - `_GLIBCXX_BITS_SF_LEGENDRE_TCC`, 438
- `sf_mod_bessel.tcc`
 - `_GLIBCXX_BITS_SF_MOD_BESSEL_TCC`, 440
- `sf_owens.t.tcc`
 - `_GLIBCXX_BITS_SF_OWENS_T_TCC`, 441
- `sf_polylog.tcc`
 - `_GLIBCXX_BITS_SF_POLYLOG_TCC`, 444
- `sf_stirling.tcc`
 - `_GLIBCXX_BITS_SF_STIRLING_TCC`, 445
- `sf_theta.tcc`
 - `_GLIBCXX_BITS_SF_THETA_TCC`, 447
- `sf_trig.tcc`
 - `_GLIBCXX_BITS_SF_TRIG_TCC`, 449
- `sf_trigint.tcc`
 - `_GLIBCXX_BITS_SF_TRIGINT_TCC`, 451
- `sf_zeta.tcc`
 - `_GLIBCXX_BITS_SF_ZETA_TCC`, 453
- `sin_pi`
 - GNU Extended Mathematical Special Functions, 133
- `sin_pif`
 - GNU Extended Mathematical Special Functions, 133
- `sin_pil`
 - GNU Extended Mathematical Special Functions, 133
- `sinc`
 - GNU Extended Mathematical Special Functions, 133
- `sinc_pi`
 - GNU Extended Mathematical Special Functions, 134
- `sinc_pif`
 - GNU Extended Mathematical Special Functions, 134
- `sinc_pil`
 - GNU Extended Mathematical Special Functions, 134
- GNU Extended Mathematical Special Functions, 134
- `sincf`
 - GNU Extended Mathematical Special Functions, 135
- `sincl`
 - GNU Extended Mathematical Special Functions, 135
- `sincos`
 - GNU Extended Mathematical Special Functions, 135
- `sincos_pi`
 - GNU Extended Mathematical Special Functions, 136
- `sincos_pif`
 - GNU Extended Mathematical Special Functions, 136
- `sincos_pil`
 - GNU Extended Mathematical Special Functions, 136
- `sincosf`
 - GNU Extended Mathematical Special Functions, 136
- `sincosl`
 - GNU Extended Mathematical Special Functions, 136
- `sinh_pi`
 - GNU Extended Mathematical Special Functions, 137
- `sinh_pif`
 - GNU Extended Mathematical Special Functions, 137
- `sinh_pil`
 - GNU Extended Mathematical Special Functions, 137
- `sinhc`
 - GNU Extended Mathematical Special Functions, 137
- `sinhc_pi`
 - GNU Extended Mathematical Special Functions, 138
- `sinhc_pif`
 - GNU Extended Mathematical Special Functions, 138
- `sinhc_pil`
 - GNU Extended Mathematical Special Functions, 138
- `sinhcf`
 - GNU Extended Mathematical Special Functions, 139
- `sinhcl`
 - GNU Extended Mathematical Special Functions, 139
- `sinhint`
 - GNU Extended Mathematical Special Functions, 139
- `sinhintf`
 - GNU Extended Mathematical Special Functions, 140
- `sinhintl`
 - GNU Extended Mathematical Special Functions, 140
- `sinint`
 - GNU Extended Mathematical Special Functions, 140
- `sinintf`
 - GNU Extended Mathematical Special Functions, 140
- `sinintl`
 - GNU Extended Mathematical Special Functions, 141
- `specfun.h`
 - `_STDCPP_MATH_SPEC_FUNCS__`, 469
 - `__cpp_lib_math_special_functions`, 469
- `sph_bessel`
 - C++17/IS29124 Mathematical Special Functions, 41
- `sph_bessel_i`
 - GNU Extended Mathematical Special Functions, 141

- sph_bessel_if
 - GNU Extended Mathematical Special Functions, [142](#)
- sph_bessel_il
 - GNU Extended Mathematical Special Functions, [142](#)
- sph_bessel_k
 - GNU Extended Mathematical Special Functions, [142](#)
- sph_bessel_kf
 - GNU Extended Mathematical Special Functions, [143](#)
- sph_bessel_kl
 - GNU Extended Mathematical Special Functions, [143](#)
- sph_besself
 - C++17/IS29124 Mathematical Special Functions, [42](#)
- sph_bessell
 - C++17/IS29124 Mathematical Special Functions, [42](#)
- sph_hankel_1
 - GNU Extended Mathematical Special Functions, [143](#), [144](#)
- sph_hankel_1f
 - GNU Extended Mathematical Special Functions, [144](#)
- sph_hankel_1l
 - GNU Extended Mathematical Special Functions, [144](#), [145](#)
- sph_hankel_2
 - GNU Extended Mathematical Special Functions, [145](#), [146](#)
- sph_hankel_2f
 - GNU Extended Mathematical Special Functions, [146](#)
- sph_hankel_2l
 - GNU Extended Mathematical Special Functions, [146](#), [147](#)
- sph_harmonic
 - GNU Extended Mathematical Special Functions, [147](#)
- sph_harmonicf
 - GNU Extended Mathematical Special Functions, [147](#)
- sph_harmonicl
 - GNU Extended Mathematical Special Functions, [148](#)
- sph_legendre
 - C++17/IS29124 Mathematical Special Functions, [42](#)
- sph_legendref
 - C++17/IS29124 Mathematical Special Functions, [43](#)
- sph_legendrel
 - C++17/IS29124 Mathematical Special Functions, [43](#)
- sph_neumann
 - C++17/IS29124 Mathematical Special Functions, [43](#)
- sph_neumannf
 - C++17/IS29124 Mathematical Special Functions, [44](#)
- sph_neumannl
 - C++17/IS29124 Mathematical Special Functions, [44](#)
- std, [182](#)
- std::__detail, [184](#)
 - _Num_Euler_Maclaurin_zeta, [329](#)
 - _S_Euler_Maclaurin_zeta, [330](#)
 - _S_double_factorial_table, [330](#)
 - _S_factorial_table, [330](#)
 - _S_harmonic_denom, [330](#)
 - _S_harmonic_number, [330](#)
 - _S_neg_double_factorial_table, [330](#)
 - _S_num_double_factorials, [331](#)
 - _S_num_double_factorials< double >, [331](#)
 - _S_num_double_factorials< float >, [331](#)
 - _S_num_double_factorials< long double >, [331](#)
 - _S_num_factorials, [331](#)
 - _S_num_factorials< double >, [331](#)
 - _S_num_factorials< float >, [331](#)
 - _S_num_factorials< long double >, [331](#)
 - _S_num_harmonic_number, [331](#)
 - _S_num_neg_double_factorials, [332](#)
 - _S_num_neg_double_factorials< double >, [332](#)
 - _S_num_neg_double_factorials< float >, [332](#)
 - _S_num_neg_double_factorials< long double >, [332](#)
 - _S_num_zetam1, [332](#)
 - _S_zetam1, [332](#)
 - __airy, [206](#)
 - __airy_ai, [206](#)
 - __airy_arg, [206](#)
 - __airy_bi, [207](#)
 - __assoc_laguerre, [207](#)
 - __assoc_legendre_p, [208](#)
 - __bernoulli, [208](#), [209](#)
 - __bernoulli_2n, [209](#)
 - __bernoulli_series, [210](#)
 - __beta, [210](#)
 - __beta_gamma, [211](#)
 - __beta_inc, [211](#)
 - __beta_lgamma, [212](#)
 - __beta_product, [212](#)
 - __binomial, [213](#)
 - __binomial_cdf, [214](#)
 - __binomial_cdfc, [214](#)
 - __binomial_pdf, [215](#)
 - __bose_einstein, [215](#)
 - __chebyshev_recur, [216](#)
 - __chebyshev_t, [216](#)
 - __chebyshev_u, [217](#)
 - __chebyshev_v, [218](#)
 - __chebyshev_w, [218](#)
 - __chi_squared_pdf, [219](#)
 - __chi_squared_pdfc, [219](#)
 - __chshint, [219](#)
 - __chshint_cont_frac, [219](#)
 - __chshint_series, [220](#)
 - __clamp_0_m2pi, [220](#)
 - __clamp_pi, [220](#)
 - __clausen, [220](#), [221](#)
 - __clausen_cl, [221](#), [222](#)
 - __clausen_sl, [222](#), [223](#)
 - __comp_ellint_1, [223](#)
 - __comp_ellint_2, [224](#)

- [__comp_ellint_3](#), 224
- [__comp_ellint_d](#), 225
- [__comp_ellint_rf](#), 225
- [__comp_ellint_rg](#), 225
- [__conf_hyperg](#), 225
- [__conf_hyperg_lim](#), 226
- [__conf_hyperg_lim_series](#), 226
- [__conf_hyperg_luke](#), 227
- [__conf_hyperg_series](#), 227
- [__cos_pi](#), 227, 228
- [__cosh_pi](#), 228
- [__coshint](#), 228
- [__cyl_bessel](#), 229
- [__cyl_bessel_i](#), 229
- [__cyl_bessel_ij_series](#), 230
- [__cyl_bessel_ik](#), 230
- [__cyl_bessel_ik_asymp](#), 231
- [__cyl_bessel_ik_steel](#), 231
- [__cyl_bessel_j](#), 232
- [__cyl_bessel_jn](#), 232
- [__cyl_bessel_jn_asymp](#), 232
- [__cyl_bessel_jn_neg_arg](#), 233
- [__cyl_bessel_jn_steel](#), 233
- [__cyl_bessel_k](#), 233
- [__cyl_hankel_1](#), 234
- [__cyl_hankel_2](#), 235
- [__cyl_neumann](#), 236
- [__cyl_neumann_n](#), 236
- [__dawson](#), 237
- [__dawson_cont_frac](#), 237
- [__dawson_series](#), 237
- [__debye](#), 237
- [__debye_region](#), 238
- [__dilog](#), 238
- [__dirichlet_beta](#), 239
- [__dirichlet_eta](#), 240
- [__dirichlet_lambda](#), 241
- [__double_factorial](#), 241
- [__ellint_1](#), 242
- [__ellint_2](#), 242
- [__ellint_3](#), 243
- [__ellint_cel](#), 243
- [__ellint_d](#), 243
- [__ellint_el1](#), 244
- [__ellint_el2](#), 244
- [__ellint_el3](#), 244
- [__ellint_rc](#), 244
- [__ellint_rd](#), 245
- [__ellint_rf](#), 246
- [__ellint_rg](#), 246
- [__ellint_rj](#), 247
- [__ellnome](#), 248
- [__ellnome_k](#), 248
- [__ellnome_series](#), 248
- [__euler](#), 248, 249
- [__euler_series](#), 249
- [__eulerian_1_recur](#), 249
- [__eulerian_2](#), 249
- [__eulerian_2_recur](#), 250
- [__expint](#), 250
- [__expint_E1](#), 251
- [__expint_E1_asymp](#), 251
- [__expint_E1_series](#), 252
- [__expint_Ei](#), 252
- [__expint_Ei_asymp](#), 253
- [__expint_Ei_series](#), 253
- [__expint_En_asymp](#), 254
- [__expint_En_cont_frac](#), 254
- [__expint_En_large_n](#), 255
- [__expint_En_recursion](#), 255
- [__expint_En_series](#), 256
- [__exponential_cdf](#), 256
- [__exponential_cdfc](#), 257
- [__exponential_pdf](#), 257
- [__factorial](#), 257
- [__falling_factorial](#), 257, 258
- [__fermi_dirac](#), 258
- [__fisher_f_cdf](#), 259
- [__fisher_f_cdfc](#), 259
- [__fisher_f_pdf](#), 259
- [__fock_airy](#), 260
- [__fresnel](#), 260
- [__fresnel_cont_frac](#), 261
- [__fresnel_series](#), 261
- [__gamma](#), 261, 262
- [__gamma_cdf](#), 262
- [__gamma_cdfc](#), 262
- [__gamma_cont_frac](#), 262
- [__gamma_pdf](#), 263
- [__gamma_reciprocal](#), 263
- [__gamma_reciprocal_series](#), 263
- [__gamma_series](#), 264
- [__gamma_temme](#), 264
- [__gauss](#), 266
- [__gegenbauer_poly](#), 266
- [__hankel](#), 266
- [__hankel_debye](#), 267
- [__hankel_params](#), 267
- [__hankel_uniform](#), 268
- [__hankel_uniform_olver](#), 268
- [__hankel_uniform_outer](#), 268
- [__hankel_uniform_sum](#), 269
- [__harmonic_number](#), 270
- [__hermite_zeros](#), 270
- [__heuman_lambda](#), 270
- [__hurwitz_zeta](#), 270
- [__hurwitz_zeta_euler_maclaurin](#), 271
- [__hurwitz_zeta_polylog](#), 271

- __hydrogen, 271
- __hyperg, 272
- __hyperg_luke, 272
- __hyperg_reflect, 272
- __hyperg_series, 273
- __ibeta_cont_frac, 274
- __jacobi_sncndn, 274
- __jacobi_zeta, 274
- __laguerre, 274
- __laguerre_zeros, 275
- __lanczos_binet1p, 275
- __lanczos_log_gamma1p, 275
- __legendre_q, 276
- __legendre_zeros, 276
- __log_binomial, 276, 277
- __log_binomial_sign, 278
- __log_double_factorial, 278
- __log_factorial, 279
- __log_falling_factorial, 279
- __log_gamma, 279, 280
- __log_gamma_bernoulli, 280
- __log_gamma_sign, 281
- __log_rising_factorial, 281
- __log_stirling_1, 281
- __log_stirling_1_sign, 282
- __log_stirling_2, 282
- __logint, 282
- __logistic_cdf, 282
- __logistic_pdf, 283
- __lognormal_cdf, 283
- __lognormal_pdf, 283
- __max_FGH, 329
- __max_FGH< double >, 329
- __max_FGH< float >, 329
- __normal_cdf, 283
- __normal_pdf, 284
- __owens_t, 284
- __pgamma, 285
- __polar_pi, 285
- __poly_hermite, 285
- __poly_hermite_asymp, 286
- __poly_hermite_recursion, 286
- __poly_jacobi, 287
- __poly_laguerre, 287
- __poly_laguerre_hyperg, 288
- __poly_laguerre_large_n, 289
- __poly_laguerre_recursion, 289
- __poly_legendre_p, 290
- __poly_prob_hermite_recursion, 291
- __poly_radial_jacobi, 291
- __polylog, 292, 293
- __polylog_exp, 293
- __polylog_exp_asymp, 294
- __polylog_exp_neg, 294, 295
- __polylog_exp_neg_int, 295, 296
- __polylog_exp_neg_real, 296, 297
- __polylog_exp_pos, 297, 298
- __polylog_exp_pos_int, 299, 300
- __polylog_exp_pos_real, 300
- __polylog_exp_sum, 301
- __psi, 301, 302
- __psi_asymp, 302
- __psi_series, 303
- __qgamma, 303
- __rice_pdf, 303
- __riemann_zeta, 304
- __riemann_zeta_euler_maclaurin, 304
- __riemann_zeta_glob, 304
- __riemann_zeta_m_1, 305
- __riemann_zeta_m_1_glob, 305
- __riemann_zeta_product, 305
- __riemann_zeta_sum, 306
- __rising_factorial, 306, 307
- __sin_pi, 307
- __sinc, 308
- __sinc_pi, 308
- __sincos, 308
- __sincos_pi, 309
- __sincosint, 309
- __sincosint_asymp, 309
- __sincosint_cont_frac, 309
- __sincosint_series, 309
- __sinh_pi, 310
- __sinhc, 310
- __sinhc_pi, 310
- __sinhint, 310
- __sph_bessel, 311
- __sph_bessel_ik, 312
- __sph_bessel_jn, 312
- __sph_bessel_jn_neg_arg, 313
- __sph_hankel, 313
- __sph_hankel_1, 313, 314
- __sph_hankel_2, 314, 315
- __sph_harmonic, 315
- __sph_legendre, 316
- __sph_neumann, 316, 317
- __spouge_binet1p, 317
- __spouge_log_gamma1p, 318
- __stirling_1, 319
- __stirling_1_recur, 319
- __stirling_1_series, 319
- __stirling_2, 319
- __stirling_2_recur, 320
- __stirling_2_series, 320
- __student_t_cdf, 320
- __student_t_cdfc, 321
- __student_t_pdf, 321
- __tan_pi, 321, 322

- __tanh_pi, [322](#)
- __tgamma, [322](#)
- __tgamma_lower, [323](#)
- __theta_1, [323](#)
- __theta_2, [323](#)
- __theta_2_asyp, [324](#)
- __theta_2_sum, [324](#)
- __theta_3, [324](#)
- __theta_3_asyp, [325](#)
- __theta_3_sum, [325](#)
- __theta_4, [325](#)
- __theta_c, [326](#)
- __theta_d, [326](#)
- __theta_n, [326](#)
- __theta_s, [326](#)
- __tricoli_u, [326](#)
- __tricoli_u_naive, [327](#)
- __weibull_cdf, [327](#)
- __weibull_pdf, [328](#)
- __zernike, [328](#)
- __znorm1, [329](#)
- __znorm2, [329](#)
- std::__detail::__gamma_lanczos_data< _Tp >, [359](#)
- std::__detail::__gamma_lanczos_data< double >, [359](#)
 - _S_cheby, [359](#)
 - _S_g, [359](#)
- std::__detail::__gamma_lanczos_data< float >, [360](#)
 - _S_cheby, [360](#)
 - _S_g, [360](#)
- std::__detail::__gamma_lanczos_data< long double >, [361](#)
 - _S_cheby, [361](#)
 - _S_g, [361](#)
- std::__detail::__gamma_spouge_data< _Tp >, [362](#)
- std::__detail::__gamma_spouge_data< double >, [362](#)
 - _S_cheby, [362](#)
- std::__detail::__gamma_spouge_data< float >, [363](#)
 - _S_cheby, [363](#)
- std::__detail::__gamma_spouge_data< long double >, [363](#)
 - _S_cheby, [364](#)
- std::__detail::__Airy
 - _Airy, [365](#)
 - inner_radius, [366](#)
 - operator(), [365](#)
 - outer_radius, [366](#)
 - scalar_type, [365](#)
 - value_type, [365](#)
- std::__detail::__Airy< _Tp >, [364](#)
- std::__detail::__Airy_asyp
 - _Airy_asyp, [368](#)
 - _Cmplx, [368](#)
 - _S_absarg_ge_pio3, [368](#)
 - _S_absarg_lt_pio3, [368](#)
 - operator(), [369](#)
- std::__detail::__Airy_asyp< _Tp >, [366](#)
- std::__detail::__Airy_asyp_data< _Tp >, [370](#)
- std::__detail::__Airy_asyp_data< double >, [370](#)
 - _S_c, [371](#)
 - _S_d, [371](#)
 - _S_max_cd, [371](#)
- std::__detail::__Airy_asyp_data< float >, [371](#)
 - _S_c, [372](#)
 - _S_d, [372](#)
 - _S_max_cd, [372](#)
- std::__detail::__Airy_asyp_data< long double >, [372](#)
 - _S_c, [372](#)
 - _S_d, [372](#)
 - _S_max_cd, [373](#)
- std::__detail::__Airy_asyp_series
 - _Airy_asyp_series, [374](#)
 - _S_sqrt_pi, [375](#)
 - operator(), [374](#)
 - scalar_type, [374](#)
 - value_type, [374](#)
- std::__detail::__Airy_asyp_series< _Sum >, [373](#)
- std::__detail::__Airy_default_radii< _Tp >, [375](#)
- std::__detail::__Airy_default_radii< double >, [375](#)
 - inner_radius, [376](#)
 - outer_radius, [376](#)
- std::__detail::__Airy_default_radii< float >, [376](#)
 - inner_radius, [376](#)
 - outer_radius, [376](#)
- std::__detail::__Airy_default_radii< long double >, [377](#)
 - inner_radius, [377](#)
 - outer_radius, [377](#)
- std::__detail::__Airy_series
 - _Cmplx, [378](#)
 - _N_FGH, [382](#)
 - _S_Ai, [379](#)
 - _S_Ai0, [382](#)
 - _S_Aip0, [382](#)
 - _S_Airy, [379](#)
 - _S_Bi, [379](#)
 - _S_Bi0, [382](#)
 - _S_Bip0, [382](#)
 - _S_FGH, [380](#)
 - _S_Fock, [380](#)
 - _S_Gi0, [383](#)
 - _S_Gip0, [383](#)
 - _S_Hi0, [383](#)
 - _S_Hip0, [383](#)
 - _S_Scorer, [381](#)
 - _S_Scorer2, [381](#)
 - _S_eps, [383](#)
 - _S_i, [383](#)
 - _S_pi, [383](#)
 - _S_sqrt_pi, [383](#)

- GNU Extended Mathematical Special Functions, [160](#)
- theta_nl
 - GNU Extended Mathematical Special Functions, [160](#)
- theta_s
 - GNU Extended Mathematical Special Functions, [160](#)
- theta_sf
 - GNU Extended Mathematical Special Functions, [161](#)
- theta_sl
 - GNU Extended Mathematical Special Functions, [161](#)
- tricoli_u
 - GNU Extended Mathematical Special Functions, [161](#)
- tricoli_uf
 - GNU Extended Mathematical Special Functions, [162](#)
- tricoli_ul
 - GNU Extended Mathematical Special Functions, [162](#)
- true_Wronskian
 - std::__detail::_AiryState, [386](#)
- value_type
 - std::__detail::_Airy, [365](#)
 - std::__detail::_Airy_asymp_series, [374](#)
- weibull_cdf
 - GNU Extended Mathematical Special Functions, [162](#)
- weibull_pdf
 - GNU Extended Mathematical Special Functions, [162](#)
- Wronskian
 - std::__detail::_AiryState, [386](#)
- zernike
 - GNU Extended Mathematical Special Functions, [163](#)
- zernikef
 - GNU Extended Mathematical Special Functions, [163](#)
- zernikel
 - GNU Extended Mathematical Special Functions, [164](#)