

C++ Special Math Functions

2.0

Generated by Doxygen 1.8.13

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	6
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()	22
8.2.2.2	assoc_laguerref()	23
8.2.2.3	assoc_laguerrel()	23
8.2.2.4	assoc_legendre()	24
8.2.2.5	assoc_legendref()	24
8.2.2.6	assoc_legendrel()	25
8.2.2.7	beta()	25
8.2.2.8	betaf()	26
8.2.2.9	betal()	26
8.2.2.10	comp_ellint_1()	27
8.2.2.11	comp_ellint_1f()	27
8.2.2.12	comp_ellint_1l()	28
8.2.2.13	comp_ellint_2()	28
8.2.2.14	comp_ellint_2f()	29
8.2.2.15	comp_ellint_2l()	29
8.2.2.16	comp_ellint_3()	29
8.2.2.17	comp_ellint_3f()	30
8.2.2.18	comp_ellint_3l()	30
8.2.2.19	cyl_bessel_i()	31

8.2.2.20	cyl_bessel_if()	31
8.2.2.21	cyl_bessel_il()	32
8.2.2.22	cyl_bessel_jl()	32
8.2.2.23	cyl_bessel_jf()	33
8.2.2.24	cyl_bessel_jl()	33
8.2.2.25	cyl_bessel_k()	33
8.2.2.26	cyl_bessel_kf()	34
8.2.2.27	cyl_bessel_kl()	34
8.2.2.28	cyl_neumann()	35
8.2.2.29	cyl_neumannf()	35
8.2.2.30	cyl_neumannl()	36
8.2.2.31	ellint_1()	36
8.2.2.32	ellint_1f()	37
8.2.2.33	ellint_1l()	37
8.2.2.34	ellint_2()	37
8.2.2.35	ellint_2f()	38
8.2.2.36	ellint_2l()	38
8.2.2.37	ellint_3()	39
8.2.2.38	ellint_3f()	40
8.2.2.39	ellint_3l()	40
8.2.2.40	expint()	40
8.2.2.41	expintf()	41
8.2.2.42	expintl()	41
8.2.2.43	hermite()	42
8.2.2.44	hermitef()	42
8.2.2.45	hermitel()	43
8.2.2.46	laguerre()	43
8.2.2.47	laguerref()	44

8.2.2.48	laguerrel()	44
8.2.2.49	legendre()	44
8.2.2.50	legendref()	45
8.2.2.51	legendrel()	45
8.2.2.52	riemann_zeta()	46
8.2.2.53	riemann_zetaf()	46
8.2.2.54	riemann_zetal()	47
8.2.2.55	sph_bessel()	47
8.2.2.56	sph_besself()	48
8.2.2.57	sph_bessell()	48
8.2.2.58	sph_legendre()	48
8.2.2.59	sph_legendref()	49
8.2.2.60	sph_legendrel()	49
8.2.2.61	sph_neumann()	50
8.2.2.62	sph_neumannf()	50
8.2.2.63	sph_neumannl()	51
8.3	GNU Extended Mathematical Special Functions	52
8.3.1	Detailed Description	64
8.3.2	Function Documentation	65
8.3.2.1	airy_ai() ^[1/2]	65
8.3.2.2	airy_ai() ^[2/2]	65
8.3.2.3	airy_aif()	66
8.3.2.4	airy_ail()	66
8.3.2.5	airy_bi() ^[1/2]	66
8.3.2.6	airy_bi() ^[2/2]	67
8.3.2.7	airy_bif()	67
8.3.2.8	airy_bil()	68
8.3.2.9	bernoulli() ^[1/2]	68

8.3.2.10	bernoulli() [2/2]	69
8.3.2.11	bernoullif()	69
8.3.2.12	bernoullil()	70
8.3.2.13	binomial()	70
8.3.2.14	binomial_p()	71
8.3.2.15	binomial_pdf()	71
8.3.2.16	binomialf()	72
8.3.2.17	binomiall()	72
8.3.2.18	bose_einstein()	72
8.3.2.19	bose_einsteinf()	73
8.3.2.20	bose_einsteinl()	73
8.3.2.21	chebyshev_t()	74
8.3.2.22	chebyshev_tf()	75
8.3.2.23	chebyshev_tl()	75
8.3.2.24	chebyshev_u()	76
8.3.2.25	chebyshev_uf()	76
8.3.2.26	chebyshev_ul()	77
8.3.2.27	chebyshev_v()	77
8.3.2.28	chebyshev_vf()	78
8.3.2.29	chebyshev_vl()	78
8.3.2.30	chebyshev_w()	78
8.3.2.31	chebyshev_wf()	79
8.3.2.32	chebyshev_wl()	79
8.3.2.33	clausen() [1/2]	80
8.3.2.34	clausen() [2/2]	80
8.3.2.35	clausen_cl()	81
8.3.2.36	clausen_clf()	81
8.3.2.37	clausen_cll()	82

8.3.2.38	<code>clausen_sl()</code>	82
8.3.2.39	<code>clausen_slf()</code>	83
8.3.2.40	<code>clausen_sll()</code>	83
8.3.2.41	<code>clausenf()</code> [1/2]	83
8.3.2.42	<code>clausenf()</code> [2/2]	84
8.3.2.43	<code>clausenl()</code> [1/2]	84
8.3.2.44	<code>clausenl()</code> [2/2]	84
8.3.2.45	<code>comp_ellint_d()</code>	84
8.3.2.46	<code>comp_ellint_df()</code>	85
8.3.2.47	<code>comp_ellint_dl()</code>	85
8.3.2.48	<code>comp_ellint_rf()</code> [1/3]	86
8.3.2.49	<code>comp_ellint_rf()</code> [2/3]	86
8.3.2.50	<code>comp_ellint_rf()</code> [3/3]	86
8.3.2.51	<code>comp_ellint_rg()</code> [1/3]	87
8.3.2.52	<code>comp_ellint_rg()</code> [2/3]	87
8.3.2.53	<code>comp_ellint_rg()</code> [3/3]	87
8.3.2.54	<code>conf_hyperg()</code>	88
8.3.2.55	<code>conf_hyperg_lim()</code>	88
8.3.2.56	<code>conf_hyperg_limf()</code>	89
8.3.2.57	<code>conf_hyperg_liml()</code>	89
8.3.2.58	<code>conf_hypergf()</code>	90
8.3.2.59	<code>conf_hypergl()</code>	90
8.3.2.60	<code>cos_pi()</code>	90
8.3.2.61	<code>cos_pif()</code>	91
8.3.2.62	<code>cos_pil()</code>	91
8.3.2.63	<code>cosh_pi()</code>	91
8.3.2.64	<code>cosh_pif()</code>	92
8.3.2.65	<code>cosh_pil()</code>	92

8.3.2.66	<code>coshint()</code>	92
8.3.2.67	<code>coshintf()</code>	93
8.3.2.68	<code>coshintl()</code>	93
8.3.2.69	<code>cosint()</code>	93
8.3.2.70	<code>cosintf()</code>	94
8.3.2.71	<code>cosintl()</code>	94
8.3.2.72	<code>cyl_hankel_1()</code> [1/2]	94
8.3.2.73	<code>cyl_hankel_1()</code> [2/2]	95
8.3.2.74	<code>cyl_hankel_1f()</code> [1/2]	95
8.3.2.75	<code>cyl_hankel_1f()</code> [2/2]	96
8.3.2.76	<code>cyl_hankel_1l()</code> [1/2]	96
8.3.2.77	<code>cyl_hankel_1l()</code> [2/2]	97
8.3.2.78	<code>cyl_hankel_2()</code> [1/2]	97
8.3.2.79	<code>cyl_hankel_2()</code> [2/2]	98
8.3.2.80	<code>cyl_hankel_2f()</code> [1/2]	98
8.3.2.81	<code>cyl_hankel_2f()</code> [2/2]	99
8.3.2.82	<code>cyl_hankel_2l()</code> [1/2]	99
8.3.2.83	<code>cyl_hankel_2l()</code> [2/2]	99
8.3.2.84	<code>dawson()</code>	100
8.3.2.85	<code>dawsonf()</code>	100
8.3.2.86	<code>dawsonl()</code>	100
8.3.2.87	<code>debye()</code>	101
8.3.2.88	<code>debyef()</code>	101
8.3.2.89	<code>debyel()</code>	102
8.3.2.90	<code>digamma()</code>	102
8.3.2.91	<code>digammaf()</code>	102
8.3.2.92	<code>digammal()</code>	103
8.3.2.93	<code>dilog()</code>	103

8.3.2.94	<code>dilogf()</code>	104
8.3.2.95	<code>dilogl()</code>	104
8.3.2.96	<code>dirichlet_beta()</code>	104
8.3.2.97	<code>dirichlet_betaf()</code>	105
8.3.2.98	<code>dirichlet_betall()</code>	105
8.3.2.99	<code>dirichlet_eta()</code>	105
8.3.2.100	<code>dirichlet_etaf()</code>	106
8.3.2.101	<code>dirichlet_etall()</code>	106
8.3.2.102	<code>dirichlet_lambda()</code>	106
8.3.2.103	<code>dirichlet_lambdaf()</code>	107
8.3.2.104	<code>dirichlet_lambdall()</code>	107
8.3.2.105	<code>double_factorial()</code>	107
8.3.2.106	<code>double_factorialf()</code>	108
8.3.2.107	<code>double_factoriall()</code>	108
8.3.2.108	<code>ellint_cel()</code>	108
8.3.2.109	<code>ellint_celf()</code>	109
8.3.2.110	<code>ellint_cell()</code>	109
8.3.2.111	<code>ellint_d()</code>	110
8.3.2.112	<code>ellint_df()</code>	110
8.3.2.113	<code>ellint_dl()</code>	110
8.3.2.114	<code>ellint_el1()</code>	111
8.3.2.115	<code>ellint_el1f()</code>	111
8.3.2.116	<code>ellint_el1l()</code>	111
8.3.2.117	<code>ellint_el2()</code>	112
8.3.2.118	<code>ellint_el2f()</code>	112
8.3.2.119	<code>ellint_el2l()</code>	113
8.3.2.120	<code>ellint_el3()</code>	113
8.3.2.121	<code>ellint_el3f()</code>	114

8.3.2.122 <code>ellint_el3l()</code>	114
8.3.2.123 <code>ellint_rc()</code>	114
8.3.2.124 <code>ellint_rcf()</code>	115
8.3.2.125 <code>ellint_rcl()</code>	115
8.3.2.126 <code>ellint_rd()</code>	116
8.3.2.127 <code>ellint_rdf()</code>	116
8.3.2.128 <code>ellint_rdl()</code>	117
8.3.2.129 <code>ellint_rf()</code>	117
8.3.2.130 <code>ellint_rff()</code>	117
8.3.2.131 <code>ellint_rfl()</code>	118
8.3.2.132 <code>ellint_rg()</code>	118
8.3.2.133 <code>ellint_rgf()</code>	119
8.3.2.134 <code>ellint_rgl()</code>	119
8.3.2.135 <code>ellint_rj()</code>	120
8.3.2.136 <code>ellint_rjf()</code>	120
8.3.2.137 <code>ellint_rjl()</code>	121
8.3.2.138 <code>ellnome()</code>	121
8.3.2.139 <code>ellnomef()</code>	122
8.3.2.140 <code>ellnomel()</code>	122
8.3.2.141 <code>euler()</code>	122
8.3.2.142 <code>eulerian_1()</code>	123
8.3.2.143 <code>eulerian_2()</code>	123
8.3.2.144 <code>expint()</code>	124
8.3.2.145 <code>expintf()</code>	124
8.3.2.146 <code>expintl()</code>	125
8.3.2.147 <code>exponential_p()</code>	125
8.3.2.148 <code>exponential_pdf()</code>	125
8.3.2.149 <code>factorial()</code>	126

8.3.2.150 factorialf()	126
8.3.2.151 factoriall()	126
8.3.2.152 falling_factorial()	127
8.3.2.153 falling_factorialf()	127
8.3.2.154 falling_factoriall()	127
8.3.2.155 fermi_dirac()	128
8.3.2.156 fermi_diracf()	128
8.3.2.157 fermi_diracl()	129
8.3.2.158 fisher_f_p()	129
8.3.2.159 fisher_f_pdf()	129
8.3.2.160 fresnel_c()	130
8.3.2.161 fresnel_cf()	130
8.3.2.162 fresnel_cl()	131
8.3.2.163 fresnel_s()	131
8.3.2.164 fresnel_sf()	131
8.3.2.165 fresnel_sl()	132
8.3.2.166 gamma_p()	132
8.3.2.167 gamma_pdf()	132
8.3.2.168 gamma_pf()	133
8.3.2.169 gamma_pl()	133
8.3.2.170 gamma_q()	133
8.3.2.171 gamma_qf()	133
8.3.2.172 gamma_ql()	134
8.3.2.173 gamma_reciprocal()	134
8.3.2.174 gamma_reciprocalf()	134
8.3.2.175 gamma_reciprocall()	135
8.3.2.176 gegenbauer()	135
8.3.2.177 gegenbauerf()	136

8.3.2.178 gegenbauerl()	136
8.3.2.179 harmonic()	136
8.3.2.180 heuman_lambda()	137
8.3.2.181 heuman_lambdaf()	137
8.3.2.182 heuman_lambdal()	138
8.3.2.183 hurwitz_zeta() $[1/2]$	138
8.3.2.184 hurwitz_zeta() $[2/2]$	138
8.3.2.185 hurwitz_zetaf()	139
8.3.2.186 hurwitz_zetal()	139
8.3.2.187 hyperg()	139
8.3.2.188 hypergf()	140
8.3.2.189 hypergl()	140
8.3.2.190 ibeta()	141
8.3.2.191 ibetac()	141
8.3.2.192 ibetacf()	142
8.3.2.193 ibetacl()	142
8.3.2.194 ibetaf()	142
8.3.2.195 ibetal()	143
8.3.2.196 jacobi()	143
8.3.2.197 jacobi_cn()	144
8.3.2.198 jacobi_cnf()	144
8.3.2.199 jacobi_cnl()	145
8.3.2.200 jacobi_dn()	145
8.3.2.201 jacobi_dnf()	146
8.3.2.202 jacobi_dnl()	146
8.3.2.203 jacobi_sn()	147
8.3.2.204 jacobi_snf()	147
8.3.2.205 jacobi_snl()	148

8.3.2.206 jacobi_theta_1()	148
8.3.2.207 jacobi_theta_1f()	148
8.3.2.208 jacobi_theta_1l()	149
8.3.2.209 jacobi_theta_2()	149
8.3.2.210 jacobi_theta_2f()	150
8.3.2.211 jacobi_theta_2l()	150
8.3.2.212 jacobi_theta_3()	150
8.3.2.213 jacobi_theta_3f()	151
8.3.2.214 jacobi_theta_3l()	151
8.3.2.215 jacobi_theta_4()	151
8.3.2.216 jacobi_theta_4f()	152
8.3.2.217 jacobi_theta_4l()	152
8.3.2.218 jacobi_zeta()	152
8.3.2.219 jacobi_zetaf()	153
8.3.2.220 jacobi_zetal()	153
8.3.2.221 jacobif()	153
8.3.2.222 jacobil()	154
8.3.2.223 lbinomial()	154
8.3.2.224 lbinomialf()	155
8.3.2.225 lbinomiall()	155
8.3.2.226 ldouble_factorial()	155
8.3.2.227 ldouble_factorialf()	156
8.3.2.228 ldouble_factoriall()	156
8.3.2.229 legendre_q()	156
8.3.2.230 legendre_qf()	157
8.3.2.231 legendre_ql()	157
8.3.2.232 lerch_phi()	158
8.3.2.233 lerch_phif()	158

8.3.2.234 lerch_phil()	158
8.3.2.235 lfactorial()	159
8.3.2.236 lfactorialf()	159
8.3.2.237 lfactoriall()	159
8.3.2.238 lfalling_factorial()	160
8.3.2.239 lfalling_factorialf()	160
8.3.2.240 lfalling_factoriall()	160
8.3.2.241 lgamma() ^[1/2]	161
8.3.2.242 lgamma() ^[2/2]	161
8.3.2.243 lgammaf() ^[1/2]	161
8.3.2.244 lgammaf() ^[2/2]	162
8.3.2.245 lgammal() ^[1/2]	162
8.3.2.246 lgammal() ^[2/2]	162
8.3.2.247 logint()	162
8.3.2.248 logintf()	163
8.3.2.249 logintl()	163
8.3.2.250 logistic_p()	164
8.3.2.251 logistic_pdf()	164
8.3.2.252 lognormal_p()	164
8.3.2.253 lognormal_pdf()	165
8.3.2.254 lrising_factorial()	165
8.3.2.255 lrising_factorialf()	165
8.3.2.256 lrising_factoriall()	166
8.3.2.257 normal_p()	166
8.3.2.258 normal_pdf()	166
8.3.2.259 owens_t()	167
8.3.2.260 owens_tf()	167
8.3.2.261 owens_tl()	167

8.3.2.262 polygamma()	168
8.3.2.263 polygammaf()	168
8.3.2.264 polygammal()	168
8.3.2.265 polylog() ^[1/2]	169
8.3.2.266 polylog() ^[2/2]	169
8.3.2.267 polylogf() ^[1/2]	170
8.3.2.268 polylogf() ^[2/2]	170
8.3.2.269 polylogl() ^[1/2]	170
8.3.2.270 polylogl() ^[2/2]	171
8.3.2.271 radpoly()	171
8.3.2.272 radpolyf()	172
8.3.2.273 radpolyl()	172
8.3.2.274 rising_factorial()	173
8.3.2.275 rising_factorialf()	173
8.3.2.276 rising_factoriall()	173
8.3.2.277 sin_pi()	174
8.3.2.278 sin_pif()	174
8.3.2.279 sin_pil()	174
8.3.2.280 sinc()	175
8.3.2.281 sinc_pi()	175
8.3.2.282 sinc_pif()	176
8.3.2.283 sinc_pil()	176
8.3.2.284 sincf()	176
8.3.2.285 sincl()	177
8.3.2.286 sincos() ^[1/2]	177
8.3.2.287 sincos() ^[2/2]	177
8.3.2.288 sincos_pi()	178
8.3.2.289 sincos_pif()	178

8.3.2.290 sincos_pil()	178
8.3.2.291 sincosf()	179
8.3.2.292 sincosl()	179
8.3.2.293 sinh_pi()	179
8.3.2.294 sinh_pif()	180
8.3.2.295 sinh_pil()	180
8.3.2.296 sinhc()	180
8.3.2.297 sinhc_pi()	181
8.3.2.298 sinhc_pif()	181
8.3.2.299 sinhc_pil()	182
8.3.2.300 sinhcf()	182
8.3.2.301 sinhcl()	182
8.3.2.302 sinhint()	183
8.3.2.303 sinhintf()	183
8.3.2.304 sinhintl()	183
8.3.2.305 sinint()	184
8.3.2.306 sinintf()	184
8.3.2.307 sinintl()	184
8.3.2.308 sph_bessel_i()	185
8.3.2.309 sph_bessel_if()	185
8.3.2.310 sph_bessel_il()	186
8.3.2.311 sph_bessel_k()	186
8.3.2.312 sph_bessel_kf()	187
8.3.2.313 sph_bessel_kl()	187
8.3.2.314 sph_hankel_1() ^[1/2]	187
8.3.2.315 sph_hankel_1() ^[2/2]	188
8.3.2.316 sph_hankel_1f() ^[1/2]	188
8.3.2.317 sph_hankel_1f() ^[2/2]	189

8.3.2.318 sph_hankel_1l() [1/2]	189
8.3.2.319 sph_hankel_1l() [2/2]	190
8.3.2.320 sph_hankel_2() [1/2]	190
8.3.2.321 sph_hankel_2() [2/2]	191
8.3.2.322 sph_hankel_2f() [1/2]	191
8.3.2.323 sph_hankel_2f() [2/2]	192
8.3.2.324 sph_hankel_2l() [1/2]	192
8.3.2.325 sph_hankel_2l() [2/2]	192
8.3.2.326 sph_harmonic()	193
8.3.2.327 sph_harmonicf()	193
8.3.2.328 sph_harmonicl()	194
8.3.2.329 stirling_1()	194
8.3.2.330 stirling_2()	195
8.3.2.331 student_t_p()	195
8.3.2.332 student_t_pdf()	196
8.3.2.333 tan_pi()	196
8.3.2.334 tan_pif()	197
8.3.2.335 tan_pil()	197
8.3.2.336 tanh_pi()	197
8.3.2.337 tanh_pif()	198
8.3.2.338 tanh_pil()	198
8.3.2.339 tgamma() [1/3]	198
8.3.2.340 tgamma() [2/3]	199
8.3.2.341 tgamma() [3/3]	199
8.3.2.342 tgamma_lower()	199
8.3.2.343 tgamma_lowerf()	200
8.3.2.344 tgamma_lowerl()	200
8.3.2.345 tgammaf() [1/3]	200

8.3.2.346 tgammaf() [2/3]	201
8.3.2.347 tgammaf() [3/3]	201
8.3.2.348 tgammaf() [1/3]	201
8.3.2.349 tgammaf() [2/3]	202
8.3.2.350 tgammaf() [3/3]	202
8.3.2.351 theta_1()	202
8.3.2.352 theta_1f()	203
8.3.2.353 theta_1l()	203
8.3.2.354 theta_2()	203
8.3.2.355 theta_2f()	204
8.3.2.356 theta_2l()	204
8.3.2.357 theta_3()	204
8.3.2.358 theta_3f()	205
8.3.2.359 theta_3l()	205
8.3.2.360 theta_4()	205
8.3.2.361 theta_4f()	206
8.3.2.362 theta_4l()	206
8.3.2.363 theta_c()	207
8.3.2.364 theta_cf()	207
8.3.2.365 theta_cl()	208
8.3.2.366 theta_d()	208
8.3.2.367 theta_df()	209
8.3.2.368 theta_dl()	209
8.3.2.369 theta_n()	209
8.3.2.370 theta_nf()	210
8.3.2.371 theta_nl()	210
8.3.2.372 theta_s()	211
8.3.2.373 theta_sf()	211
8.3.2.374 theta_sl()	212
8.3.2.375 tricomi_u()	212
8.3.2.376 tricomi_uf()	213
8.3.2.377 tricomi_ul()	213
8.3.2.378 weibull_p()	213
8.3.2.379 weibull_pdf()	214
8.3.2.380 zernike()	214
8.3.2.381 zernikef()	215
8.3.2.382 zernikel()	215

9 Namespace Documentation	217
9.1 <code>__gnu_cxx</code> Namespace Reference	217
9.1.1 Enumeration Type Documentation	231
9.1.1.1 <code>gauss_quad_type</code>	231
9.1.2 Function Documentation	231
9.1.2.1 <code>__fp_is_equal()</code>	231
9.1.2.2 <code>__fp_is_even_integer()</code>	232
9.1.2.3 <code>__fp_is_half_integer()</code>	232
9.1.2.4 <code>__fp_is_half_odd_integer()</code>	233
9.1.2.5 <code>__fp_is_integer()</code>	233
9.1.2.6 <code>__fp_is_odd_integer()</code>	234
9.1.2.7 <code>__fp_is_zero()</code>	234
9.1.2.8 <code>__fp_max_abs()</code>	235
9.1.2.9 <code>__parity()</code>	235
9.2 <code>std</code> Namespace Reference	236
9.3 <code>std::__detail</code> Namespace Reference	238
9.3.1 Detailed Description	261
9.3.2 Function Documentation	262
9.3.2.1 <code>__airy()</code>	262
9.3.2.2 <code>__airy_ai()</code>	262
9.3.2.3 <code>__airy_arg()</code>	262
9.3.2.4 <code>__airy_bi()</code>	263
9.3.2.5 <code>__assoc_laguerre()</code>	263
9.3.2.6 <code>__assoc_legendre_p()</code>	264
9.3.2.7 <code>__bernoulli()</code> ^[1/2]	265
9.3.2.8 <code>__bernoulli()</code> ^[2/2]	265
9.3.2.9 <code>__bernoulli_2n()</code>	266
9.3.2.10 <code>__bernoulli_series()</code>	266

9.3.2.11	<code>__beta()</code>	267
9.3.2.12	<code>__beta_gamma()</code>	267
9.3.2.13	<code>__beta_inc()</code>	268
9.3.2.14	<code>__beta_lgamma()</code>	269
9.3.2.15	<code>__beta_p()</code>	269
9.3.2.16	<code>__beta_product()</code>	270
9.3.2.17	<code>__binomial()</code> $[1/2]$	270
9.3.2.18	<code>__binomial()</code> $[2/2]$	271
9.3.2.19	<code>__binomial_p()</code>	272
9.3.2.20	<code>__binomial_pdf()</code>	272
9.3.2.21	<code>__binomial_q()</code>	273
9.3.2.22	<code>__bose_einstein()</code>	273
9.3.2.23	<code>__cauchy_p()</code>	274
9.3.2.24	<code>__chebyshev_recur()</code>	274
9.3.2.25	<code>__chebyshev_t()</code>	275
9.3.2.26	<code>__chebyshev_u()</code>	276
9.3.2.27	<code>__chebyshev_v()</code>	276
9.3.2.28	<code>__chebyshev_w()</code>	277
9.3.2.29	<code>__chi_squared_pdf()</code>	278
9.3.2.30	<code>__chi_squared_pdfc()</code>	278
9.3.2.31	<code>__chshint()</code>	279
9.3.2.32	<code>__chshint_cont_frac()</code>	279
9.3.2.33	<code>__chshint_series()</code>	279
9.3.2.34	<code>__clamp_0_m2pi()</code>	280
9.3.2.35	<code>__clamp_pi()</code>	280
9.3.2.36	<code>__clausen()</code> $[1/2]$	280
9.3.2.37	<code>__clausen()</code> $[2/2]$	281
9.3.2.38	<code>__clausen_cl()</code> $[1/2]$	281

9.3.2.39	__clausen_cl() [2/2]	282
9.3.2.40	__clausen_sl() [1/2]	283
9.3.2.41	__clausen_sl() [2/2]	283
9.3.2.42	__comp_ellint_1()	284
9.3.2.43	__comp_ellint_2()	284
9.3.2.44	__comp_ellint_3()	286
9.3.2.45	__comp_ellint_d()	287
9.3.2.46	__comp_ellint_rf()	287
9.3.2.47	__comp_ellint_rg()	287
9.3.2.48	__conf_hyperg()	287
9.3.2.49	__conf_hyperg_lim()	288
9.3.2.50	__conf_hyperg_lim_series()	289
9.3.2.51	__conf_hyperg_luke()	289
9.3.2.52	__conf_hyperg_series()	290
9.3.2.53	__cos_pi() [1/2]	290
9.3.2.54	__cos_pi() [2/2]	291
9.3.2.55	__cosh_pi() [1/2]	291
9.3.2.56	__cosh_pi() [2/2]	291
9.3.2.57	__coshint()	291
9.3.2.58	__coulomb_CF1()	292
9.3.2.59	__coulomb_CF2()	292
9.3.2.60	__coulomb_f_recur()	293
9.3.2.61	__coulomb_g_recur()	293
9.3.2.62	__coulomb_norm()	293
9.3.2.63	__cyl_bessel()	294
9.3.2.64	__cyl_bessel_i()	294
9.3.2.65	__cyl_bessel_ij_series()	295
9.3.2.66	__cyl_bessel_ik()	296

9.3.2.67	__cyl_bessel_ik_asymp()	296
9.3.2.68	__cyl_bessel_ik_steel()	297
9.3.2.69	__cyl_bessel_j()	297
9.3.2.70	__cyl_bessel_jn()	298
9.3.2.71	__cyl_bessel_jn_asymp()	298
9.3.2.72	__cyl_bessel_jn_neg_arg()	299
9.3.2.73	__cyl_bessel_jn_steel()	299
9.3.2.74	__cyl_bessel_k()	300
9.3.2.75	__cyl_hankel_1() $[1/2]$	301
9.3.2.76	__cyl_hankel_1() $[2/2]$	301
9.3.2.77	__cyl_hankel_2() $[1/2]$	302
9.3.2.78	__cyl_hankel_2() $[2/2]$	302
9.3.2.79	__cyl_neumann()	303
9.3.2.80	__cyl_neumann_n()	303
9.3.2.81	__dawson()	304
9.3.2.82	__dawson_cont_frac()	304
9.3.2.83	__dawson_series()	305
9.3.2.84	__debye()	305
9.3.2.85	__debye_region()	306
9.3.2.86	__digamma() $[1/2]$	306
9.3.2.87	__digamma() $[2/2]$	307
9.3.2.88	__digamma_asymp()	307
9.3.2.89	__digamma_series()	308
9.3.2.90	__dilog()	308
9.3.2.91	__dirichlet_beta() $[1/2]$	308
9.3.2.92	__dirichlet_beta() $[2/2]$	309
9.3.2.93	__dirichlet_eta() $[1/2]$	310
9.3.2.94	__dirichlet_eta() $[2/2]$	310

9.3.2.95	<code>__dirichlet_lambda()</code>	311
9.3.2.96	<code>__double_factorial()</code>	311
9.3.2.97	<code>__ellint_1()</code>	312
9.3.2.98	<code>__ellint_2()</code>	313
9.3.2.99	<code>__ellint_3()</code>	313
9.3.2.100	<code>__ellint_cel()</code>	314
9.3.2.101	<code>__ellint_d()</code>	314
9.3.2.102	<code>__ellint_el1()</code>	314
9.3.2.103	<code>__ellint_el2()</code>	315
9.3.2.104	<code>__ellint_el3()</code>	315
9.3.2.105	<code>__ellint_rc()</code>	315
9.3.2.106	<code>__ellint_rd()</code>	316
9.3.2.107	<code>__ellint_rf()</code>	317
9.3.2.108	<code>__ellint_rg()</code>	318
9.3.2.109	<code>__ellint_rj()</code>	318
9.3.2.110	<code>__ellnome()</code>	319
9.3.2.111	<code>__ellnome_k()</code>	320
9.3.2.112	<code>__ellnome_series()</code>	320
9.3.2.113	<code>__euler()</code> $[1/2]$	320
9.3.2.114	<code>__euler()</code> $[2/2]$	321
9.3.2.115	<code>__euler_series()</code>	321
9.3.2.116	<code>__eulerian_1()</code>	322
9.3.2.117	<code>__eulerian_1_recur()</code>	322
9.3.2.118	<code>__eulerian_2()</code>	322
9.3.2.119	<code>__eulerian_2_recur()</code>	323
9.3.2.120	<code>__exp2()</code>	323
9.3.2.121	<code>__expint()</code> $[1/2]$	323
9.3.2.122	<code>__expint()</code> $[2/2]$	324

9.3.2.123	<code>__expint_E1()</code>	325
9.3.2.124	<code>__expint_E1_asymp()</code>	325
9.3.2.125	<code>__expint_E1_series()</code>	326
9.3.2.126	<code>__expint_Ei()</code>	326
9.3.2.127	<code>__expint_Ei_asymp()</code>	327
9.3.2.128	<code>__expint_Ei_series()</code>	328
9.3.2.129	<code>__expint_En_asymp()</code>	328
9.3.2.130	<code>__expint_En_cont_frac()</code>	329
9.3.2.131	<code>__expint_En_large_n()</code>	329
9.3.2.132	<code>__expint_En_recursion()</code>	330
9.3.2.133	<code>__expint_En_series()</code>	331
9.3.2.134	<code>__exponential_p()</code>	331
9.3.2.135	<code>__exponential_pdf()</code>	332
9.3.2.136	<code>__exponential_q()</code>	332
9.3.2.137	<code>__factorial()</code>	332
9.3.2.138	<code>__falling_factorial()</code> $[1/2]$	333
9.3.2.139	<code>__falling_factorial()</code> $[2/2]$	333
9.3.2.140	<code>__fermi_dirac()</code>	333
9.3.2.141	<code>__fisher_f_p()</code>	334
9.3.2.142	<code>__fisher_f_pdf()</code>	335
9.3.2.143	<code>__fisher_f_q()</code>	335
9.3.2.144	<code>__fock_airy()</code>	336
9.3.2.145	<code>__fresnel()</code>	336
9.3.2.146	<code>__fresnel_cont_frac()</code>	337
9.3.2.147	<code>__fresnel_series()</code>	337
9.3.2.148	<code>__gamma()</code> $[1/2]$	337
9.3.2.149	<code>__gamma()</code> $[2/2]$	338
9.3.2.150	<code>__gamma_cont_frac()</code>	338

9.3.2.151	<code>__gamma_p()</code> [1/2]	339
9.3.2.152	<code>__gamma_p()</code> [2/2]	339
9.3.2.153	<code>__gamma_pdf()</code>	340
9.3.2.154	<code>__gamma_q()</code> [1/2]	340
9.3.2.155	<code>__gamma_q()</code> [2/2]	341
9.3.2.156	<code>__gamma_reciprocal()</code>	341
9.3.2.157	<code>__gamma_reciprocal_series()</code>	342
9.3.2.158	<code>__gamma_series()</code>	342
9.3.2.159	<code>__gamma_temme()</code>	343
9.3.2.160	<code>__gauss()</code>	343
9.3.2.161	<code>__gegenbauer_recur()</code>	344
9.3.2.162	<code>__gegenbauer_zeros()</code>	344
9.3.2.163	<code>__hankel()</code>	344
9.3.2.164	<code>__hankel_debye()</code>	345
9.3.2.165	<code>__hankel_params()</code>	346
9.3.2.166	<code>__hankel_uniform()</code>	346
9.3.2.167	<code>__hankel_uniform_olver()</code>	347
9.3.2.168	<code>__hankel_uniform_outer()</code>	347
9.3.2.169	<code>__hankel_uniform_sum()</code>	348
9.3.2.170	<code>__harmonic_number()</code>	349
9.3.2.171	<code>__hermite()</code>	349
9.3.2.172	<code>__hermite_asymp()</code>	350
9.3.2.173	<code>__hermite_recur()</code>	351
9.3.2.174	<code>__hermite_zeros()</code>	352
9.3.2.175	<code>__heuman_lambda()</code>	352
9.3.2.176	<code>__hurwitz_zeta()</code>	352
9.3.2.177	<code>__hurwitz_zeta_euler_maclaurin()</code>	353
9.3.2.178	<code>__hurwitz_zeta_polylog()</code>	353

9.3.2.179	__hydrogen()	354
9.3.2.180	__hyperg()	355
9.3.2.181	__hyperg_luke()	355
9.3.2.182	__hyperg_recur()	356
9.3.2.183	__hyperg_reflect()	356
9.3.2.184	__hyperg_series()	357
9.3.2.185	__ibeta_cont_frac()	358
9.3.2.186	__jacobi_ellint()	358
9.3.2.187	__jacobi_recur()	359
9.3.2.188	__jacobi_theta_1() [1/2]	359
9.3.2.189	__jacobi_theta_1() [2/2]	360
9.3.2.190	__jacobi_theta_1_prod()	361
9.3.2.191	__jacobi_theta_1_sum()	361
9.3.2.192	__jacobi_theta_2() [1/2]	362
9.3.2.193	__jacobi_theta_2() [2/2]	363
9.3.2.194	__jacobi_theta_2_prod()	364
9.3.2.195	__jacobi_theta_2_sum()	364
9.3.2.196	__jacobi_theta_3() [1/2]	365
9.3.2.197	__jacobi_theta_3() [2/2]	366
9.3.2.198	__jacobi_theta_3_prod()	366
9.3.2.199	__jacobi_theta_3_sum()	367
9.3.2.200	__jacobi_theta_4() [1/2]	367
9.3.2.201	__jacobi_theta_4() [2/2]	368
9.3.2.202	__jacobi_theta_4_prod()	369
9.3.2.203	__jacobi_theta_4_sum()	369
9.3.2.204	__jacobi_zeros()	370
9.3.2.205	__jacobi_zeta()	370
9.3.2.206	__kolmogorov_p()	371

9.3.2.207 <code>__laguerre()</code> [1/2]	371
9.3.2.208 <code>__laguerre()</code> [2/2]	372
9.3.2.209 <code>__laguerre_hyperg()</code>	373
9.3.2.210 <code>__laguerre_large_n()</code>	373
9.3.2.211 <code>__laguerre_recur()</code>	374
9.3.2.212 <code>__laguerre_zeros()</code>	375
9.3.2.213 <code>__lanczos_binet1p()</code>	375
9.3.2.214 <code>__lanczos_log_gamma1p()</code>	376
9.3.2.215 <code>__legendre_p()</code>	377
9.3.2.216 <code>__legendre_q()</code>	377
9.3.2.217 <code>__legendre_zeros()</code>	378
9.3.2.218 <code>__log_binomial()</code> [1/2]	378
9.3.2.219 <code>__log_binomial()</code> [2/2]	379
9.3.2.220 <code>__log_binomial_sign()</code> [1/2]	379
9.3.2.221 <code>__log_binomial_sign()</code> [2/2]	380
9.3.2.222 <code>__log_double_factorial()</code> [1/2]	380
9.3.2.223 <code>__log_double_factorial()</code> [2/2]	381
9.3.2.224 <code>__log_factorial()</code>	381
9.3.2.225 <code>__log_falling_factorial()</code>	382
9.3.2.226 <code>__log_gamma()</code> [1/2]	382
9.3.2.227 <code>__log_gamma()</code> [2/2]	383
9.3.2.228 <code>__log_gamma_bernoulli()</code>	383
9.3.2.229 <code>__log_gamma_sign()</code> [1/2]	384
9.3.2.230 <code>__log_gamma_sign()</code> [2/2]	384
9.3.2.231 <code>__log_rising_factorial()</code>	385
9.3.2.232 <code>__log_stirling_1()</code>	385
9.3.2.233 <code>__log_stirling_1_sign()</code>	385
9.3.2.234 <code>__log_stirling_2()</code>	386

9.3.2.235	__logint()	386
9.3.2.236	__logistic_p()	386
9.3.2.237	__logistic_pdf()	387
9.3.2.238	__lognormal_p()	387
9.3.2.239	__lognormal_pdf()	388
9.3.2.240	__normal_p()	388
9.3.2.241	__normal_pdf()	388
9.3.2.242	__owens_t()	389
9.3.2.243	__polar_pi() ^[1/2]	389
9.3.2.244	__polar_pi() ^[2/2]	390
9.3.2.245	__polygamma()	390
9.3.2.246	__polylog() ^[1/2]	390
9.3.2.247	__polylog() ^[2/2]	391
9.3.2.248	__polylog_exp()	392
9.3.2.249	__polylog_exp_asymp()	392
9.3.2.250	__polylog_exp_neg() ^[1/2]	393
9.3.2.251	__polylog_exp_neg() ^[2/2]	394
9.3.2.252	__polylog_exp_neg_int() ^[1/2]	394
9.3.2.253	__polylog_exp_neg_int() ^[2/2]	395
9.3.2.254	__polylog_exp_neg_real() ^[1/2]	396
9.3.2.255	__polylog_exp_neg_real() ^[2/2]	396
9.3.2.256	__polylog_exp_pos() ^[1/3]	397
9.3.2.257	__polylog_exp_pos() ^[2/3]	398
9.3.2.258	__polylog_exp_pos() ^[3/3]	398
9.3.2.259	__polylog_exp_pos_int() ^[1/2]	399
9.3.2.260	__polylog_exp_pos_int() ^[2/2]	400
9.3.2.261	__polylog_exp_pos_real() ^[1/2]	400
9.3.2.262	__polylog_exp_pos_real() ^[2/2]	401

9.3.2.263	<code>__polylog_exp_sum()</code>	401
9.3.2.264	<code>__prob_hermite_recur()</code>	402
9.3.2.265	<code>__radial_jacobi()</code>	403
9.3.2.266	<code>__radial_jacobi_zeros()</code>	404
9.3.2.267	<code>__rice_pdf()</code>	405
9.3.2.268	<code>__riemann_zeta()</code>	405
9.3.2.269	<code>__riemann_zeta_euler_maclaurin()</code>	406
9.3.2.270	<code>__riemann_zeta_glob()</code>	406
9.3.2.271	<code>__riemann_zeta_laurent()</code>	406
9.3.2.272	<code>__riemann_zeta_m_1()</code>	407
9.3.2.273	<code>__riemann_zeta_m_1_glob()</code>	407
9.3.2.274	<code>__riemann_zeta_product()</code>	408
9.3.2.275	<code>__riemann_zeta_sum()</code>	408
9.3.2.276	<code>__rising_factorial()</code> $[1/2]$	409
9.3.2.277	<code>__rising_factorial()</code> $[2/2]$	409
9.3.2.278	<code>__sin_pi()</code> $[1/2]$	410
9.3.2.279	<code>__sin_pi()</code> $[2/2]$	410
9.3.2.280	<code>__sinc()</code>	410
9.3.2.281	<code>__sinc_pi()</code>	411
9.3.2.282	<code>__sincos()</code> $[1/4]$	411
9.3.2.283	<code>__sincos()</code> $[2/4]$	411
9.3.2.284	<code>__sincos()</code> $[3/4]$	411
9.3.2.285	<code>__sincos()</code> $[4/4]$	412
9.3.2.286	<code>__sincos_pi()</code>	412
9.3.2.287	<code>__sincosint()</code>	412
9.3.2.288	<code>__sincosint_asymp()</code>	413
9.3.2.289	<code>__sincosint_cont_frac()</code>	413
9.3.2.290	<code>__sincosint_series()</code>	413

9.3.2.291	<code>__sinh_pi()</code> [1/2]	414
9.3.2.292	<code>__sinh_pi()</code> [2/2]	414
9.3.2.293	<code>__sinhc()</code>	414
9.3.2.294	<code>__sinhc_pi()</code>	415
9.3.2.295	<code>__sinhint()</code>	415
9.3.2.296	<code>__sph_bessel()</code> [1/2]	416
9.3.2.297	<code>__sph_bessel()</code> [2/2]	416
9.3.2.298	<code>__sph_bessel_ik()</code>	417
9.3.2.299	<code>__sph_bessel_jn()</code>	417
9.3.2.300	<code>__sph_bessel_jn_neg_arg()</code>	418
9.3.2.301	<code>__sph_hankel()</code>	418
9.3.2.302	<code>__sph_hankel_1()</code> [1/2]	419
9.3.2.303	<code>__sph_hankel_1()</code> [2/2]	420
9.3.2.304	<code>__sph_hankel_2()</code> [1/2]	420
9.3.2.305	<code>__sph_hankel_2()</code> [2/2]	421
9.3.2.306	<code>__sph_harmonic()</code>	421
9.3.2.307	<code>__sph_legendre()</code>	422
9.3.2.308	<code>__sph_neumann()</code> [1/2]	423
9.3.2.309	<code>__sph_neumann()</code> [2/2]	423
9.3.2.310	<code>__spouge_binet1p()</code>	424
9.3.2.311	<code>__spouge_log_gamma1p()</code>	425
9.3.2.312	<code>__stirling_1()</code>	426
9.3.2.313	<code>__stirling_1_recur()</code>	426
9.3.2.314	<code>__stirling_1_series()</code>	427
9.3.2.315	<code>__stirling_2()</code>	427
9.3.2.316	<code>__stirling_2_recur()</code>	427
9.3.2.317	<code>__stirling_2_series()</code>	428
9.3.2.318	<code>__student_t_p()</code>	428

9.3.2.319	__student_t_pdf()	429
9.3.2.320	__student_t_q()	429
9.3.2.321	__tan_pi() [1/2]	430
9.3.2.322	__tan_pi() [2/2]	430
9.3.2.323	__tanh_pi() [1/2]	430
9.3.2.324	__tanh_pi() [2/2]	431
9.3.2.325	__tgamma()	431
9.3.2.326	__tgamma_lower()	431
9.3.2.327	__theta_1()	432
9.3.2.328	__theta_2()	432
9.3.2.329	__theta_2_asymp()	433
9.3.2.330	__theta_2_sum()	433
9.3.2.331	__theta_3()	433
9.3.2.332	__theta_3_asymp()	434
9.3.2.333	__theta_3_sum()	434
9.3.2.334	__theta_4()	435
9.3.2.335	__theta_c()	435
9.3.2.336	__theta_d()	436
9.3.2.337	__theta_n()	436
9.3.2.338	__theta_s()	436
9.3.2.339	__tricomi_u()	437
9.3.2.340	__tricomi_u_naive()	437
9.3.2.341	__weibull_p()	438
9.3.2.342	__weibull_pdf()	438
9.3.2.343	__zernike()	439
9.3.2.344	__znorm1()	439
9.3.2.345	__znorm2()	440
9.3.3	Variable Documentation	440

9.3.3.1	<code>__max_FGH</code>	440
9.3.3.2	<code>__max_FGH< double ></code>	440
9.3.3.3	<code>__max_FGH< float ></code>	440
9.3.3.4	<code>_Num_Euler_Maclaurin_zeta</code>	441
9.3.3.5	<code>_Num_Stieltjes</code>	441
9.3.3.6	<code>_S_double_factorial_table</code>	441
9.3.3.7	<code>_S_Euler_Maclaurin_zeta</code>	441
9.3.3.8	<code>_S_factorial_table</code>	442
9.3.3.9	<code>_S_harmonic_denom</code>	442
9.3.3.10	<code>_S_harmonic_numer</code>	442
9.3.3.11	<code>_S_neg_double_factorial_table</code>	442
9.3.3.12	<code>_S_num_double_factorials</code>	442
9.3.3.13	<code>_S_num_double_factorials< double ></code>	443
9.3.3.14	<code>_S_num_double_factorials< float ></code>	443
9.3.3.15	<code>_S_num_double_factorials< long double ></code>	443
9.3.3.16	<code>_S_num_factorials</code>	443
9.3.3.17	<code>_S_num_factorials< double ></code>	443
9.3.3.18	<code>_S_num_factorials< float ></code>	444
9.3.3.19	<code>_S_num_factorials< long double ></code>	444
9.3.3.20	<code>_S_num_harmonic_numer</code>	444
9.3.3.21	<code>_S_num_neg_double_factorials</code>	444
9.3.3.22	<code>_S_num_neg_double_factorials< double ></code>	444
9.3.3.23	<code>_S_num_neg_double_factorials< float ></code>	445
9.3.3.24	<code>_S_num_neg_double_factorials< long double ></code>	445
9.3.3.25	<code>_S_num_zetam1</code>	445
9.3.3.26	<code>_S_Stieltjes</code>	445
9.3.3.27	<code>_S_zetam1</code>	446

10 Class Documentation	447
10.1 <code>__gnu_cxx::__airy_t<_Tx,_Tp></code> Struct Template Reference	447
10.1.1 Detailed Description	447
10.1.2 Member Function Documentation	448
10.1.2.1 <code>__Wronskian()</code>	448
10.1.3 Member Data Documentation	448
10.1.3.1 <code>__Ai_deriv</code>	448
10.1.3.2 <code>__Ai_value</code>	448
10.1.3.3 <code>__Bi_deriv</code>	448
10.1.3.4 <code>__Bi_value</code>	449
10.1.3.5 <code>__x_arg</code>	449
10.2 <code>__gnu_cxx::__chebyshev_t_t<_Tp></code> Struct Template Reference	449
10.2.1 Detailed Description	450
10.2.2 Member Function Documentation	450
10.2.2.1 <code>deriv()</code>	450
10.2.2.2 <code>deriv2()</code>	450
10.2.3 Member Data Documentation	450
10.2.3.1 <code>__n</code>	450
10.2.3.2 <code>__T_n</code>	451
10.2.3.3 <code>__T_nm1</code>	451
10.2.3.4 <code>__T_nm2</code>	451
10.2.3.5 <code>__x</code>	451
10.3 <code>__gnu_cxx::__chebyshev_u_t<_Tp></code> Struct Template Reference	451
10.3.1 Detailed Description	452
10.3.2 Member Function Documentation	452
10.3.2.1 <code>deriv()</code>	452
10.3.3 Member Data Documentation	452
10.3.3.1 <code>__n</code>	452

10.3.3.2	<code>__U_n</code>	453
10.3.3.3	<code>__U_nm1</code>	453
10.3.3.4	<code>__U_nm2</code>	453
10.3.3.5	<code>__x</code>	453
10.4	<code>__gnu_cxx::__chebyshev_v_t<_Tp></code> Struct Template Reference	453
10.4.1	Detailed Description	454
10.4.2	Member Function Documentation	454
10.4.2.1	<code>deriv()</code>	454
10.4.3	Member Data Documentation	454
10.4.3.1	<code>__n</code>	454
10.4.3.2	<code>__V_n</code>	455
10.4.3.3	<code>__V_nm1</code>	455
10.4.3.4	<code>__V_nm2</code>	455
10.4.3.5	<code>__x</code>	455
10.5	<code>__gnu_cxx::__chebyshev_w_t<_Tp></code> Struct Template Reference	455
10.5.1	Detailed Description	456
10.5.2	Member Function Documentation	456
10.5.2.1	<code>deriv()</code>	456
10.5.3	Member Data Documentation	456
10.5.3.1	<code>__n</code>	456
10.5.3.2	<code>__W_n</code>	457
10.5.3.3	<code>__W_nm1</code>	457
10.5.3.4	<code>__W_nm2</code>	457
10.5.3.5	<code>__x</code>	457
10.6	<code>__gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp></code> Struct Template Reference	457
10.6.1	Detailed Description	458
10.6.2	Member Function Documentation	458
10.6.2.1	<code>__Wronskian()</code>	458

10.6.3	Member Data Documentation	459
10.6.3.1	__J_deriv	459
10.6.3.2	__J_value	459
10.6.3.3	__N_deriv	459
10.6.3.4	__N_value	459
10.6.3.5	__nu_arg	460
10.6.3.6	__x_arg	460
10.7	__gnu_cxx::__cyl_coulomb_t<_Teta, _Trho, _Tp> Struct Template Reference	460
10.7.1	Detailed Description	461
10.7.2	Member Function Documentation	461
10.7.2.1	__Wronskian()	461
10.7.3	Member Data Documentation	461
10.7.3.1	__eta_arg	461
10.7.3.2	__F_deriv	461
10.7.3.3	__F_value	462
10.7.3.4	__G_deriv	462
10.7.3.5	__G_value	462
10.7.3.6	__I	462
10.7.3.7	__rho_arg	463
10.8	__gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp> Struct Template Reference	463
10.8.1	Detailed Description	463
10.8.2	Member Function Documentation	464
10.8.2.1	__Wronskian()	464
10.8.3	Member Data Documentation	464
10.8.3.1	__H1_deriv	464
10.8.3.2	__H1_value	464
10.8.3.3	__H2_deriv	464
10.8.3.4	__H2_value	465

10.8.3.5	__nu_arg	465
10.8.3.6	__x_arg	465
10.9	__gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp> Struct Template Reference	465
10.9.1	Detailed Description	466
10.9.2	Member Function Documentation	466
10.9.2.1	__Wronskian()	466
10.9.3	Member Data Documentation	466
10.9.3.1	__I_deriv	467
10.9.3.2	__I_value	467
10.9.3.3	__K_deriv	467
10.9.3.4	__K_value	467
10.9.3.5	__nu_arg	468
10.9.3.6	__x_arg	468
10.10	__gnu_cxx::__fock_airy_t<_Tx, _Tp> Struct Template Reference	468
10.10.1	Detailed Description	469
10.10.2	Member Function Documentation	469
10.10.2.1	__Wronskian()	469
10.10.3	Member Data Documentation	469
10.10.3.1	__w1_deriv	469
10.10.3.2	__w1_value	469
10.10.3.3	__w2_deriv	470
10.10.3.4	__w2_value	470
10.10.3.5	__x_arg	470
10.11	__gnu_cxx::__fp_is_integer_t Struct Reference	470
10.11.1	Detailed Description	471
10.11.2	Member Function Documentation	471
10.11.2.1	operator bool()	471
10.11.2.2	operator()()	471

10.11.3 Member Data Documentation	471
10.11.3.1 <code>__is_integral</code>	471
10.11.3.2 <code>__value</code>	472
10.12 <code>__gnu_cxx::__gamma_inc_t<_Tp></code> Struct Template Reference	472
10.12.1 Detailed Description	472
10.12.2 Member Data Documentation	472
10.12.2.1 <code>__lgamma_value</code>	472
10.12.2.2 <code>__tgamma_value</code>	473
10.13 <code>__gnu_cxx::__gamma_temme_t<_Tp></code> Struct Template Reference	473
10.13.1 Detailed Description	474
10.13.2 Member Data Documentation	474
10.13.2.1 <code>__gamma_1_value</code>	474
10.13.2.2 <code>__gamma_2_value</code>	474
10.13.2.3 <code>__gamma_minus_value</code>	475
10.13.2.4 <code>__gamma_plus_value</code>	475
10.13.2.5 <code>__mu_arg</code>	475
10.14 <code>__gnu_cxx::__gappa_pq_t<_Tp></code> Struct Template Reference	475
10.14.1 Detailed Description	476
10.14.2 Member Data Documentation	476
10.14.2.1 <code>__gappa_p_value</code>	476
10.14.2.2 <code>__gappa_q_value</code>	476
10.15 <code>__gnu_cxx::__gegenbauer_t<_Tp></code> Struct Template Reference	476
10.15.1 Detailed Description	477
10.15.2 Member Function Documentation	477
10.15.2.1 <code>deriv()</code>	477
10.15.3 Member Data Documentation	477
10.15.3.1 <code>__C_n</code>	477
10.15.3.2 <code>__C_nm1</code>	478

10.15.3.3	__C_nm2	478
10.15.3.4	__lambda	478
10.15.3.5	__n	478
10.15.3.6	__x	478
10.16	__gnu_cxx::__hermite_he_t<_Tp> Struct Template Reference	479
10.16.1	Detailed Description	479
10.16.2	Member Function Documentation	479
10.16.2.1	deriv()	479
10.16.2.2	deriv2()	479
10.16.3	Member Data Documentation	480
10.16.3.1	__He_n	480
10.16.3.2	__He_nm1	480
10.16.3.3	__He_nm2	480
10.16.3.4	__n	480
10.16.3.5	__x	481
10.17	__gnu_cxx::__hermite_t<_Tp> Struct Template Reference	481
10.17.1	Detailed Description	481
10.17.2	Member Function Documentation	481
10.17.2.1	deriv()	482
10.17.2.2	deriv2()	482
10.17.3	Member Data Documentation	482
10.17.3.1	__H_n	482
10.17.3.2	__H_nm1	482
10.17.3.3	__H_nm2	482
10.17.3.4	__n	483
10.17.3.5	__x	483
10.18	__gnu_cxx::__jacobi_ellint_t<_Tp> Struct Template Reference	483
10.18.1	Detailed Description	484

10.18.2 Member Function Documentation	484
10.18.2.1 __am()	484
10.18.2.2 __cd()	484
10.18.2.3 __cn_deriv()	484
10.18.2.4 __cs()	484
10.18.2.5 __dc()	485
10.18.2.6 __ds()	485
10.18.2.7 __nc()	485
10.18.2.8 __nd()	485
10.18.2.9 __ns()	485
10.18.2.10 __sc()	486
10.18.2.11 __sd()	486
10.18.2.12 __sn_deriv()	486
10.18.3 Member Data Documentation	486
10.18.3.1 __cn_value	486
10.18.3.2 __dn_value	487
10.18.3.3 __sn_value	487
10.19 __gnu_cxx::__jacobi_t< _Tp > Struct Template Reference	487
10.19.1 Detailed Description	488
10.19.2 Member Function Documentation	488
10.19.2.1 deriv()	488
10.19.3 Member Data Documentation	488
10.19.3.1 __alpha1	488
10.19.3.2 __beta1	488
10.19.3.3 __n	489
10.19.3.4 __P_n	489
10.19.3.5 __P_nm1	489
10.19.3.6 __P_nm2	489

10.19.3.7 __x	489
10.20 __gnu_cxx::__laguerre_t<_Tpa, _Tp> Struct Template Reference	490
10.20.1 Detailed Description	490
10.20.2 Member Function Documentation	490
10.20.2.1 deriv()	490
10.20.3 Member Data Documentation	490
10.20.3.1 __alpha1	491
10.20.3.2 __L_n	491
10.20.3.3 __L_nm1	491
10.20.3.4 __L_nm2	491
10.20.3.5 __n	491
10.20.3.6 __x	492
10.21 __gnu_cxx::__legendre_p_t<_Tp> Struct Template Reference	492
10.21.1 Detailed Description	492
10.21.2 Member Function Documentation	492
10.21.2.1 deriv()	493
10.21.3 Member Data Documentation	493
10.21.3.1 __l	493
10.21.3.2 __P_l	493
10.21.3.3 __P_lm1	493
10.21.3.4 __P_lm2	493
10.21.3.5 __z	494
10.22 __gnu_cxx::__lgamma_t<_Tp> Struct Template Reference	494
10.22.1 Detailed Description	494
10.22.2 Member Data Documentation	494
10.22.2.1 __lgamma_sign	494
10.22.2.2 __lgamma_value	495
10.23 __gnu_cxx::__quadrature_point_t<_Tp> Struct Template Reference	495

10.23.1 Detailed Description	495
10.23.2 Constructor & Destructor Documentation	495
10.23.2.1 __quadrature_point_t() [1/2]	496
10.23.2.2 __quadrature_point_t() [2/2]	496
10.23.3 Member Data Documentation	496
10.23.3.1 __point	496
10.23.3.2 __weight	496
10.24 __gnu_cxx::__sincos_t< _Tp > Struct Template Reference	496
10.24.1 Detailed Description	497
10.24.2 Member Data Documentation	497
10.24.2.1 __cos_v	497
10.24.2.2 __sin_v	497
10.25 __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp > Struct Template Reference	498
10.25.1 Detailed Description	498
10.25.2 Member Function Documentation	498
10.25.2.1 __Wronskian()	498
10.25.3 Member Data Documentation	499
10.25.3.1 __j_deriv	499
10.25.3.2 __j_value	499
10.25.3.3 __n_arg	499
10.25.3.4 __n_deriv	499
10.25.3.5 __n_value	500
10.25.3.6 __x_arg	500
10.26 __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp > Struct Template Reference	500
10.26.1 Detailed Description	501
10.26.2 Member Function Documentation	501
10.26.2.1 __Wronskian()	501
10.26.3 Member Data Documentation	501

10.26.3.1	__h1_deriv	501
10.26.3.2	__h1_value	501
10.26.3.3	__h2_deriv	502
10.26.3.4	__h2_value	502
10.26.3.5	__n_arg	502
10.26.3.6	__x_arg	502
10.27	__gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp > Struct Template Reference	503
10.27.1	Detailed Description	503
10.27.2	Member Function Documentation	503
10.27.2.1	__Wronskian()	503
10.27.3	Member Data Documentation	504
10.27.3.1	__i_deriv	504
10.27.3.2	__i_value	504
10.27.3.3	__k_deriv	504
10.27.3.4	__k_value	504
10.27.3.5	__n_arg	505
10.27.3.6	__x_arg	505
10.28	std::__detail::__gamma_lanczos_data< _Tp > Struct Template Reference	505
10.28.1	Detailed Description	505
10.29	std::__detail::__gamma_lanczos_data< double > Struct Template Reference	505
10.29.1	Detailed Description	506
10.29.2	Member Data Documentation	506
10.29.2.1	_S_cheby	506
10.29.2.2	_S_g	506
10.30	std::__detail::__gamma_lanczos_data< float > Struct Template Reference	506
10.30.1	Detailed Description	507
10.30.2	Member Data Documentation	507
10.30.2.1	_S_cheby	507

10.30.2.2 _S_g	507
10.31 std::__detail::__gamma_lanczos_data< long double > Struct Template Reference	507
10.31.1 Detailed Description	508
10.31.2 Member Data Documentation	508
10.31.2.1 _S_cheby	508
10.31.2.2 _S_g	508
10.32 std::__detail::__gamma_spouge_data< _Tp > Struct Template Reference	509
10.32.1 Detailed Description	509
10.33 std::__detail::__gamma_spouge_data< double > Struct Template Reference	509
10.33.1 Detailed Description	509
10.33.2 Member Data Documentation	509
10.33.2.1 _S_cheby	510
10.34 std::__detail::__gamma_spouge_data< float > Struct Template Reference	510
10.34.1 Detailed Description	510
10.34.2 Member Data Documentation	510
10.34.2.1 _S_cheby	511
10.35 std::__detail::__gamma_spouge_data< long double > Struct Template Reference	511
10.35.1 Detailed Description	511
10.35.2 Member Data Documentation	511
10.35.2.1 _S_cheby	512
10.36 std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 > Struct Template Reference	512
10.36.1 Detailed Description	514
10.36.2 Member Typedef Documentation	514
10.36.2.1 _Cmplx	514
10.36.2.2 _Real	514
10.36.2.3 _Real_Omega1	514
10.36.2.4 _Real_Omega3	515
10.36.2.5 _Tp_Nome	515

10.36.3 Constructor & Destructor Documentation	515
10.36.3.1 __jacobi_lattice_t() [1/3]	515
10.36.3.2 __jacobi_lattice_t() [2/3]	515
10.36.3.3 __jacobi_lattice_t() [3/3]	516
10.36.4 Member Function Documentation	516
10.36.4.1 __ellnome()	516
10.36.4.2 __omega_1()	516
10.36.4.3 __omega_2()	517
10.36.4.4 __omega_3()	517
10.36.4.5 __reduce()	517
10.36.4.6 __tau()	518
10.36.5 Member Data Documentation	518
10.36.5.1 _M_omega_1	518
10.36.5.2 _M_omega_3	518
10.36.5.3 _S_pi	519
10.37std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__arg_t Struct Reference	519
10.37.1 Detailed Description	519
10.37.2 Member Data Documentation	519
10.37.2.1 __m	520
10.37.2.2 __n	520
10.37.2.3 __z	520
10.38std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau_t Struct Reference	520
10.38.1 Detailed Description	521
10.38.2 Constructor & Destructor Documentation	521
10.38.2.1 __tau_t()	521
10.38.3 Member Data Documentation	521
10.38.3.1 __val	521
10.39std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 > Struct Template Reference	522

10.39.1 Detailed Description	523
10.39.2 Member Typedef Documentation	523
10.39.2.1 _Cmplx	523
10.39.2.2 _Real	523
10.39.2.3 _Type	523
10.39.3 Constructor & Destructor Documentation	523
10.39.3.1 __jacobi_theta_0_t()	524
10.39.4 Member Function Documentation	524
10.39.4.1 dedekind_eta()	524
10.39.5 Member Data Documentation	524
10.39.5.1 eta_1	524
10.39.5.2 eta_2	525
10.39.5.3 eta_3	525
10.39.5.4 th1p	525
10.39.5.5 th1ppp	525
10.39.5.6 th2	525
10.39.5.7 th2pp	526
10.39.5.8 th3	526
10.39.5.9 th3pp	526
10.39.5.10th4	526
10.39.5.11th4pp	526
10.40std::__detail::__weierstrass_invariants_t<_Tp1,_Tp3> Struct Template Reference	527
10.40.1 Detailed Description	527
10.40.2 Member Typedef Documentation	528
10.40.2.1 _Cmplx	528
10.40.2.2 _Real	528
10.40.2.3 _Type	528
10.40.3 Constructor & Destructor Documentation	528

10.40.3.1 <code>__weierstrass_invariants_t()</code>	528
10.40.4 Member Function Documentation	529
10.40.4.1 <code>__delta()</code>	529
10.40.4.2 <code>__klein_j()</code>	529
10.40.5 Member Data Documentation	529
10.40.5.1 <code>__g_2</code>	529
10.40.5.2 <code>__g_3</code>	529
10.41 <code>std::__detail::__weierstrass_roots_t<_Tp1, _Tp3></code> Struct Template Reference	530
10.41.1 Detailed Description	531
10.41.2 Member Typedef Documentation	531
10.41.2.1 <code>_Cmplx</code>	531
10.41.2.2 <code>_Real</code>	531
10.41.2.3 <code>_Type</code>	531
10.41.3 Constructor & Destructor Documentation	532
10.41.3.1 <code>__weierstrass_roots_t()</code> [1/2]	532
10.41.3.2 <code>__weierstrass_roots_t()</code> [2/2]	532
10.41.4 Member Function Documentation	532
10.41.4.1 <code>__delta()</code>	533
10.41.5 Member Data Documentation	533
10.41.5.1 <code>__e1</code>	533
10.41.5.2 <code>__e2</code>	533
10.41.5.3 <code>__e3</code>	533
10.42 <code>std::__detail::__Airy<_Tp></code> Class Template Reference	534
10.42.1 Detailed Description	534
10.42.2 Member Typedef Documentation	534
10.42.2.1 <code>scalar_type</code>	534
10.42.2.2 <code>value_type</code>	535
10.42.3 Constructor & Destructor Documentation	535

10.42.3.1 <code>_Airy()</code> [1/3]	535
10.42.3.2 <code>_Airy()</code> [2/3]	535
10.42.3.3 <code>_Airy()</code> [3/3]	535
10.42.4 Member Function Documentation	535
10.42.4.1 <code>operator()()</code>	535
10.42.5 Member Data Documentation	536
10.42.5.1 <code>inner_radius</code>	536
10.42.5.2 <code>outer_radius</code>	536
10.43 <code>std::__detail::_Airy_asymp< _Tp ></code> Class Template Reference	536
10.43.1 Detailed Description	537
10.43.2 Member Typedef Documentation	538
10.43.2.1 <code>_Cmplx</code>	538
10.43.3 Constructor & Destructor Documentation	538
10.43.3.1 <code>_Airy_asymp()</code>	538
10.43.4 Member Function Documentation	538
10.43.4.1 <code>_S_absarg_ge_pio3()</code>	538
10.43.4.2 <code>_S_absarg_lt_pio3()</code>	539
10.43.4.3 <code>operator()()</code>	540
10.44 <code>std::__detail::_Airy_asymp_data< _Tp ></code> Struct Template Reference	540
10.44.1 Detailed Description	540
10.45 <code>std::__detail::_Airy_asymp_data< double ></code> Struct Template Reference	541
10.45.1 Detailed Description	541
10.45.2 Member Data Documentation	541
10.45.2.1 <code>_S_c</code>	541
10.45.2.2 <code>_S_d</code>	541
10.45.2.3 <code>_S_max_cd</code>	542
10.46 <code>std::__detail::_Airy_asymp_data< float ></code> Struct Template Reference	542
10.46.1 Detailed Description	542

10.46.2 Member Data Documentation	542
10.46.2.1 _S_c	542
10.46.2.2 _S_d	542
10.46.2.3 _S_max_cd	543
10.47std::__detail::__Airy_asymp_data< long double > Struct Template Reference	543
10.47.1 Detailed Description	543
10.47.2 Member Data Documentation	543
10.47.2.1 _S_c	543
10.47.2.2 _S_d	543
10.47.2.3 _S_max_cd	544
10.48std::__detail::__Airy_asymp_series< _Sum > Class Template Reference	544
10.48.1 Detailed Description	544
10.48.2 Member Typedef Documentation	545
10.48.2.1 scalar_type	545
10.48.2.2 value_type	545
10.48.3 Constructor & Destructor Documentation	545
10.48.3.1 __Airy_asymp_series() [1/3]	545
10.48.3.2 __Airy_asymp_series() [2/3]	545
10.48.3.3 __Airy_asymp_series() [3/3]	546
10.48.4 Member Function Documentation	546
10.48.4.1 operator>()	546
10.48.5 Member Data Documentation	546
10.48.5.1 _S_sqrt_pi	546
10.49std::__detail::__Airy_default_radII< _Tp > Struct Template Reference	547
10.49.1 Detailed Description	547
10.50std::__detail::__Airy_default_radII< double > Struct Template Reference	547
10.50.1 Detailed Description	547
10.50.2 Member Data Documentation	547

10.50.2.1 inner_radius	547
10.50.2.2 outer_radius	548
10.51std::__detail::_Airy_default_radii< float > Struct Template Reference	548
10.51.1 Detailed Description	548
10.51.2 Member Data Documentation	548
10.51.2.1 inner_radius	548
10.51.2.2 outer_radius	548
10.52std::__detail::_Airy_default_radii< long double > Struct Template Reference	549
10.52.1 Detailed Description	549
10.52.2 Member Data Documentation	549
10.52.2.1 inner_radius	549
10.52.2.2 outer_radius	549
10.53std::__detail::_Airy_series< _Tp > Class Template Reference	549
10.53.1 Detailed Description	550
10.53.2 Member Typedef Documentation	550
10.53.2.1 _Cmplx	551
10.53.3 Member Function Documentation	551
10.53.3.1 _S_Ai()	551
10.53.3.2 _S_Airy()	551
10.53.3.3 _S_Bi()	552
10.53.3.4 _S_FGH()	553
10.53.3.5 _S_Fock()	553
10.53.3.6 _S_Scorer()	554
10.53.3.7 _S_Scorer2()	554
10.53.4 Member Data Documentation	555
10.53.4.1 _N_FGH	555
10.53.4.2 _S_Ai0	555
10.53.4.3 _S_Aip0	555

10.53.4.4 _S_Bi0	556
10.53.4.5 _S_Bip0	556
10.53.4.6 _S_eps	556
10.53.4.7 _S_Gi0	556
10.53.4.8 _S_Gip0	556
10.53.4.9 _S_Hi0	557
10.53.4.10 _S_Hip0	557
10.53.4.11 _S_i	557
10.53.4.12 _S_pi	557
10.53.4.13 _S_sqrt_pi	557
10.54std::__detail::__AiryAuxilliaryState<_Tp> Struct Template Reference	558
10.54.1 Detailed Description	558
10.54.2 Member Typedef Documentation	558
10.54.2.1 _Val	558
10.54.3 Member Data Documentation	558
10.54.3.1 __fai_deriv	559
10.54.3.2 __fai_value	559
10.54.3.3 __gai_deriv	559
10.54.3.4 __gai_value	559
10.54.3.5 __hai_deriv	559
10.54.3.6 __hai_value	560
10.54.3.7 __z	560
10.55std::__detail::__AiryState<_Tp> Struct Template Reference	560
10.55.1 Detailed Description	561
10.55.2 Member Typedef Documentation	561
10.55.2.1 _Real	561
10.55.3 Member Function Documentation	561
10.55.3.1 true_Wronskian()	561

10.55.3.2 Wronskian()	561
10.55.4 Member Data Documentation	562
10.55.4.1 __Ai_deriv	562
10.55.4.2 __Ai_value	562
10.55.4.3 __Bi_deriv	562
10.55.4.4 __Bi_value	562
10.55.4.5 __z	563
10.56std::__detail::_AsympTerminator<_Tp> Class Template Reference	563
10.56.1 Detailed Description	563
10.56.2 Constructor & Destructor Documentation	563
10.56.2.1 _AsympTerminator()	564
10.56.3 Member Function Documentation	564
10.56.3.1 num_terms()	564
10.56.3.2 operator>()()	564
10.56.3.3 operator<<()	564
10.57std::__detail::_Factorial_table<_Tp> Struct Template Reference	565
10.57.1 Detailed Description	565
10.57.2 Member Data Documentation	565
10.57.2.1 __factorial	565
10.57.2.2 __log_factorial	565
10.57.2.3 __n	566
10.58std::__detail::_Terminator<_Tp> Class Template Reference	566
10.58.1 Detailed Description	566
10.58.2 Constructor & Destructor Documentation	566
10.58.2.1 _Terminator()	567
10.58.3 Member Function Documentation	567
10.58.3.1 num_terms()	567
10.58.3.2 operator>()()	567

11 File Documentation	569
11.1 include/bits/sfairy.tcc File Reference	569
11.1.1 Detailed Description	571
11.1.2 Macro Definition Documentation	571
11.1.2.1 _GLIBCXX_BITS_SF_AIRY_TCC	571
11.2 include/bits/sfbernoulli.tcc File Reference	571
11.2.1 Detailed Description	572
11.2.2 Macro Definition Documentation	572
11.2.2.1 _GLIBCXX_BITS_SF_BERNOULLI_TCC	572
11.3 include/bits/sfbessel.tcc File Reference	572
11.3.1 Detailed Description	574
11.3.2 Macro Definition Documentation	575
11.3.2.1 _GLIBCXX_BITS_SF_BESSEL_TCC	575
11.4 include/bits/sfbeta.tcc File Reference	575
11.4.1 Detailed Description	576
11.4.2 Macro Definition Documentation	576
11.4.2.1 _GLIBCXX_BITS_SF_BETA_TCC	576
11.5 include/bits/sfcardinal.tcc File Reference	577
11.5.1 Macro Definition Documentation	578
11.5.1.1 _GLIBCXX_BITS_SF_CARDINAL_TCC	578
11.6 include/bits/sfchebyshev.tcc File Reference	579
11.6.1 Detailed Description	580
11.6.2 Macro Definition Documentation	580
11.6.2.1 _GLIBCXX_BITS_SF_CHEBYSHEV_TCC	580
11.7 include/bits/sfcoulomb.tcc File Reference	580
11.7.1 Detailed Description	581
11.7.2 Macro Definition Documentation	582
11.7.2.1 _GLIBCXX_BITS_SF_COULOMB_TCC	582

11.8	include/bits/sf_dawson.tcc File Reference	582
11.8.1	Detailed Description	583
11.8.2	Macro Definition Documentation	583
11.8.2.1	_GLIBCXX_BITS_SF_DAWSON_TCC	583
11.9	include/bits/sf_distributions.tcc File Reference	584
11.9.1	Detailed Description	586
11.9.2	Macro Definition Documentation	587
11.9.2.1	_GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC	587
11.10	include/bits/sf_ellint.tcc File Reference	587
11.10.1	Detailed Description	589
11.10.2	Macro Definition Documentation	589
11.10.2.1	_GLIBCXX_BITS_SF_ELLINT_TCC	589
11.11	include/bits/sf_euler.tcc File Reference	590
11.11.1	Detailed Description	591
11.11.2	Macro Definition Documentation	591
11.11.2.1	_GLIBCXX_BITS_SF_EULER_TCC	591
11.12	include/bits/sf_expint.tcc File Reference	591
11.12.1	Detailed Description	593
11.12.2	Macro Definition Documentation	593
11.12.2.1	_GLIBCXX_BITS_SF_EXPINT_TCC	593
11.13	include/bits/sf_fresnel.tcc File Reference	594
11.13.1	Detailed Description	595
11.13.2	Macro Definition Documentation	595
11.13.2.1	_GLIBCXX_BITS_SF_FRESNEL_TCC	595
11.14	include/bits/sf_gamma.tcc File Reference	595
11.14.1	Detailed Description	603
11.14.2	Macro Definition Documentation	603
11.14.2.1	_GLIBCXX_BITS_SF_GAMMA_TCC	603

11.15include/bits/sf_gegenbauer.tcc File Reference	603
11.15.1 Detailed Description	604
11.15.2 Macro Definition Documentation	604
11.15.2.1 _GLIBCXX_BITS_SF_GEGENBAUER_TCC	605
11.16include/bits/sf_hankel.tcc File Reference	605
11.16.1 Detailed Description	607
11.16.2 Macro Definition Documentation	607
11.16.2.1 _GLIBCXX_BITS_SF_HANKEL_TCC	607
11.17include/bits/sf_hermite.tcc File Reference	608
11.17.1 Detailed Description	609
11.17.2 Macro Definition Documentation	609
11.17.2.1 _GLIBCXX_BITS_SF_HERMITE_TCC	609
11.18include/bits/sf_hyperg.tcc File Reference	609
11.18.1 Detailed Description	612
11.18.2 Macro Definition Documentation	612
11.18.2.1 _GLIBCXX_BITS_SF_HYPERG_TCC	612
11.19include/bits/sf_hypint.tcc File Reference	612
11.19.1 Detailed Description	613
11.19.2 Macro Definition Documentation	613
11.19.2.1 _GLIBCXX_BITS_SF_HYPINT_TCC	613
11.20include/bits/sf_jacobi.tcc File Reference	614
11.20.1 Detailed Description	615
11.20.2 Macro Definition Documentation	615
11.20.2.1 _GLIBCXX_BITS_SF_JACOBI_TCC	615
11.21include/bits/sf_laguerre.tcc File Reference	615
11.21.1 Detailed Description	617
11.21.2 Macro Definition Documentation	617
11.21.2.1 _GLIBCXX_BITS_SF_LAGUERRE_TCC	617

11.22include/bits/sf_legendre.tcc File Reference	617
11.22.1 Detailed Description	619
11.22.2 Macro Definition Documentation	619
11.22.2.1 _GLIBCXX_BITS_SF_LEGENDRE_TCC	619
11.23include/bits/sf_mod_bessel.tcc File Reference	619
11.23.1 Detailed Description	621
11.23.2 Macro Definition Documentation	621
11.23.2.1 _GLIBCXX_BITS_SF_MOD_BESSEL_TCC	621
11.24include/bits/sf_owens_t.tcc File Reference	621
11.24.1 Detailed Description	622
11.24.2 Macro Definition Documentation	622
11.24.2.1 _GLIBCXX_BITS_SF_OWENS_T_TCC	622
11.25include/bits/sf_polylog.tcc File Reference	623
11.25.1 Detailed Description	625
11.25.2 Macro Definition Documentation	625
11.25.2.1 _GLIBCXX_BITS_SF_POLYLOG_TCC	625
11.26include/bits/sf_stirling.tcc File Reference	626
11.26.1 Detailed Description	627
11.26.2 Macro Definition Documentation	627
11.26.2.1 _GLIBCXX_BITS_SF_STIRLING_TCC	627
11.27include/bits/sf_theta.tcc File Reference	627
11.27.1 Detailed Description	630
11.27.2 Macro Definition Documentation	630
11.27.2.1 _GLIBCXX_BITS_SF_THETA_TCC	630
11.28include/bits/sf_trig.tcc File Reference	631
11.28.1 Detailed Description	632
11.28.2 Macro Definition Documentation	632
11.28.2.1 _GLIBCXX_BITS_SF_TRIG_TCC	633

11.29include/bits/sf_trigint.tcc File Reference	633
11.29.1 Detailed Description	634
11.29.2 Macro Definition Documentation	634
11.29.2.1 _GLIBCXX_BITS_SF_TRIGINT_TCC	634
11.30include/bits/sf_zeta.tcc File Reference	635
11.30.1 Detailed Description	637
11.30.2 Macro Definition Documentation	637
11.30.2.1 _GLIBCXX_BITS_SF_ZETA_TCC	637
11.31include/bits/specfun.h File Reference	637
11.31.1 Detailed Description	652
11.31.2 Macro Definition Documentation	652
11.31.2.1 __cpp_lib_math_special_functions	653
11.31.2.2 __STDCPP_MATH_SPEC_FUNCS__	653
11.32include/bits/specfun_state.h File Reference	653
11.32.1 Detailed Description	655
11.33include/ext/math_util.h File Reference	655
11.33.1 Detailed Description	655

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
- [digamma](#) - Digamma or psi function
- [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
- [theta_1](#) - Jacobi theta function 1
- [theta_2](#) - Jacobi theta function 2
- [theta_3](#) - Jacobi theta function 3
- [theta_4](#) - Jacobi theta function 4
- [jacobi_zeta](#) - Jacobi zeta functions
- [lbinomial](#) - Log binomial coefficients
- [ldouble_factorial](#) - Log double factorials
- [legendre_q](#) - Legendre functions of the second kind
- [lerch](#) - The Lerch transcendent
- [lfactorial](#) - Log factorials
- [lfalling_factorial](#) - Log falling factorials
- [lgamma](#) - Log gamma for complex arguments
- [lrising_factorial](#) - Log rising factorials
- [owens_t](#) - Owens T functions
- [gamma_p](#) - Regularized lower incomplete gamma functions
- [gamma_q](#) - Regularized upper incomplete gamma functions
- [radpoly](#) - Radial polynomials
- [rising_factorial](#) - Rising factorials
- [sinhc](#) - Hyperbolic sinus cardinal function
- [sinhc_pi](#) - Reperiodized hyperbolic sinus cardinal function
- [sinc](#) - Normalized sinus cardinal function
- [sincos](#) - Sine + cosine function

- [sincos_pi](#) - Reperiodized sine + cosine function
- [sin_pi](#) - Reperiodized sine function.
- [sinh_pi](#) - Reperiodized hyperbolic sine function.
- [sinc_pi](#) - Sinus cardinal function
- [sinhint](#) - Hyperbolic sine integral
- [sinint](#) - Sine integral
- [sph_bessel_i](#) - Spherical regular modified Bessel functions
- [sph_bessel_k](#) - Spherical iregular modified Bessel functions
- [sph_hankel_1](#) - Spherical Hankel functions of the first kind
- [sph_hankel_2](#) - Spherical Hankel functions of the first kind
- [sph_harmonic](#) - Spherical
- [stirling_1](#) - Stirling numbers of the first kind
- [stirling_2](#) - Stirling numbers of the second kind
- [tan_pi](#) - Reperiodized tangent function.
- [tanh_pi](#) - Reperiodized hyperbolic tangent function.
- [tgamma](#) - Gamma for complex arguments
- [tgamma](#) - Upper incomplete gamma functions
- [tgamma_lower](#) - Lower incomplete gamma functions
- [theta_1](#) - Exponential theta function 1
- [theta_2](#) - Exponential theta function 2
- [theta_3](#) - Exponential theta function 3
- [theta_4](#) - Exponential theta function 4
- [tricomi_u](#) - Tricomi confluent hypergeometric function
- [zernike](#) - Zernike polynomials

1.3 General Features

1.3.1 Argument Promotion

The arguments 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.

Find Debye for $x < -2\pi i$

Find Debye for $x < -2\pi i$

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::__hermite_recur](#) (unsigned int __n, _Tp __x)

Find the sign of Hermite blowup values.

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	52

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

__gnu_cxx	217
std	236
std::__detail	
Implementation-space details	238

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>	447
<code>__gnu_cxx::__chebyshev_t<_Tp></code>	449
<code>__gnu_cxx::__chebyshev_u_t<_Tp></code>	451
<code>__gnu_cxx::__chebyshev_v_t<_Tp></code>	453
<code>__gnu_cxx::__chebyshev_w_t<_Tp></code>	455
<code>__gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp></code>	457
<code>__gnu_cxx::__cyl_coulomb_t<_Teta, _Trho, _Tp></code>	460
<code>__gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp></code>	463
<code>__gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp></code>	465
<code>__gnu_cxx::__fock_airy_t<_Tx, _Tp></code>	468
<code>__gnu_cxx::__fp_is_integer_t</code>	470
<code>__gnu_cxx::__gamma_inc_t<_Tp></code>	472
<code>__gnu_cxx::__gamma_temme_t<_Tp></code>	473
<code>__gnu_cxx::__gappa_pq_t<_Tp></code>	475
<code>__gnu_cxx::__gegenbauer_t<_Tp></code>	476
<code>__gnu_cxx::__hermite_he_t<_Tp></code>	479
<code>__gnu_cxx::__hermite_t<_Tp></code>	481
<code>__gnu_cxx::__jacobi_ellint_t<_Tp></code>	483
<code>__gnu_cxx::__jacobi_t<_Tp></code>	487
<code>__gnu_cxx::__laguerre_t<_Tpa, _Tp></code>	490
<code>__gnu_cxx::__legendre_p_t<_Tp></code>	492
<code>__gnu_cxx::__lgamma_t<_Tp></code>	494
<code>__gnu_cxx::__quadrature_point_t<_Tp></code>	495
<code>__gnu_cxx::__sincos_t<_Tp></code>	496
<code>__gnu_cxx::__sph_bessel_t<_Tn, _Tx, _Tp></code>	498
<code>__gnu_cxx::__sph_hankel_t<_Tn, _Tx, _Tp></code>	500
<code>__gnu_cxx::__sph_mod_bessel_t<_Tn, _Tx, _Tp></code>	503
<code>std::__detail::__jacobi_lattice_t<_Tp1, _Tp3></code>	512
<code>std::__detail::__gamma_lanczos_data<_Tp></code>	505
<code>std::__detail::__gamma_lanczos_data<double></code>	505
<code>std::__detail::__gamma_lanczos_data<float></code>	506

std::__detail::__gamma_lanczos_data< long double >	507
std::__detail::__gamma_spouge_data< _Tp >	509
std::__detail::__gamma_spouge_data< double >	509
std::__detail::__gamma_spouge_data< float >	510
std::__detail::__gamma_spouge_data< long double >	511
std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >	512
std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__arg_t	519
std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau_t	520
std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >	522
std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >	527
std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >	530
std::__detail::__Airy< _Tp >	534
std::__detail::__Airy_asymp_data< _Tp >	540
std::__detail::__Airy_asymp< _Tp >	536
std::__detail::__Airy_asymp_data< double >	541
std::__detail::__Airy_asymp_data< float >	542
std::__detail::__Airy_asymp_data< long double >	543
std::__detail::__Airy_asymp_series< _Sum >	544
std::__detail::__Airy_default_radii< _Tp >	547
std::__detail::__Airy_default_radii< double >	547
std::__detail::__Airy_default_radii< float >	548
std::__detail::__Airy_default_radii< long double >	549
std::__detail::__Airy_series< _Tp >	549
std::__detail::__AiryAuxilliaryState< _Tp >	558
std::__detail::__AiryState< _Tp >	560
std::__detail::__AsympTerminator< _Tp >	563
std::__detail::__Factorial_table< _Tp >	565
std::__detail::__Terminator< _Tp >	566

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>	447
__gnu_cxx::__chebyshev_t<_Tp>	449
__gnu_cxx::__chebyshev_u_t<_Tp>	451
__gnu_cxx::__chebyshev_v_t<_Tp>	453
__gnu_cxx::__chebyshev_w_t<_Tp>	455
__gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp>	457
__gnu_cxx::__cyl_coulomb_t<_Teta, _Trho, _Tp>	460
__gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp>	463
__gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp>	465
__gnu_cxx::__fock_airy_t<_Tx, _Tp>	468
__gnu_cxx::__fp_is_integer_t	470
__gnu_cxx::__gamma_inc_t<_Tp>	472
__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::__gappa_pq_t<_Tp>	473
__gnu_cxx::__gegenbauer_t<_Tp>	475
__gnu_cxx::__hermite_he_t<_Tp>	476
__gnu_cxx::__hermite_t<_Tp>	479
__gnu_cxx::__jacobi_ellint_t<_Tp>	481
__gnu_cxx::__jacobi_t<_Tp>	483
__gnu_cxx::__jacobi_t<_Tp>	487
__gnu_cxx::__laguerre_t<_Tpa, _Tp>	490
__gnu_cxx::__legendre_p_t<_Tp>	492

__gnu_cxx::__lgamma_t< _Tp >	494
__gnu_cxx::__quadrature_point_t< _Tp >	495
__gnu_cxx::__sincos_t< _Tp >	496
__gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >	498
__gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >	500
__gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >	503
std::__detail::__gamma_lanczos_data< _Tp >	505
std::__detail::__gamma_lanczos_data< double >	505
std::__detail::__gamma_lanczos_data< float >	506
std::__detail::__gamma_lanczos_data< long double >	507
std::__detail::__gamma_spouge_data< _Tp >	509
std::__detail::__gamma_spouge_data< double >	509
std::__detail::__gamma_spouge_data< float >	510
std::__detail::__gamma_spouge_data< long double >	511
std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >	512
std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__arg_t	519
std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau_t	520
std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >	522
std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >	527
std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >	530
std::__detail::__Airy< _Tp >	534
std::__detail::__Airy_asymp< _Tp >	536
std::__detail::__Airy_asymp_data< _Tp >	540
std::__detail::__Airy_asymp_data< double >	541
std::__detail::__Airy_asymp_data< float >	542
std::__detail::__Airy_asymp_data< long double >	543
std::__detail::__Airy_asymp_series< _Sum >	544
std::__detail::__Airy_default_radii< _Tp >	547
std::__detail::__Airy_default_radii< double >	547
std::__detail::__Airy_default_radii< float >	548
std::__detail::__Airy_default_radii< long double >	549
std::__detail::__Airy_series< _Tp >	549
std::__detail::__AiryAuxilliaryState< _Tp >	558
std::__detail::__AiryState< _Tp >	560
std::__detail::__AsympTerminator< _Tp >	563
std::__detail::__Factorial_table< _Tp >	565
std::__detail::__Terminator< _Tp >	566

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

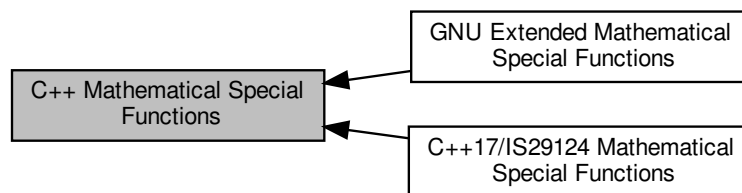
include/bits/sf_airy.tcc	569
include/bits/sf_bernoulli.tcc	571
include/bits/sf_bessel.tcc	572
include/bits/sf_beta.tcc	575
include/bits/sf_cardinal.tcc	577
include/bits/sf_chebyshev.tcc	579
include/bits/sf_coulomb.tcc	580
include/bits/sf_dawson.tcc	582
include/bits/sf_distributions.tcc	584
include/bits/sf_ellint.tcc	587
include/bits/sf_euler.tcc	590
include/bits/sf_expint.tcc	591
include/bits/sf_fresnel.tcc	594
include/bits/sf_gamma.tcc	595
include/bits/sf_gegenbauer.tcc	603
include/bits/sf_hankel.tcc	605
include/bits/sf_hermite.tcc	608
include/bits/sf_hyperg.tcc	609
include/bits/sf_hypint.tcc	612
include/bits/sf_jacobi.tcc	614
include/bits/sf_laguerre.tcc	615
include/bits/sf_legendre.tcc	617
include/bits/sf_mod_bessel.tcc	619
include/bits/sf_owens_t.tcc	621
include/bits/sf_polylog.tcc	623
include/bits/sf_stirling.tcc	626
include/bits/sf_theta.tcc	627
include/bits/sf_trig.tcc	631
include/bits/sf_trigint.tcc	633
include/bits/sf_zeta.tcc	635
include/bits/specfun.h	637
include/bits/specfun_state.h	653
include/ext/math_util.h	655

Chapter 8

Module Documentation

8.1 C++ Mathematical Special Functions

Collaboration diagram for C++ Mathematical Special Functions:



Modules

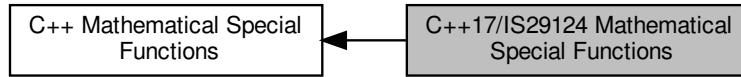
- [C++17/IS29124 Mathematical Special Functions](#)
- [GNU Extended Mathematical Special Functions](#)

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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > std::expint (_Tp __x)`
- `float std::expintf (float __x)`
- `long double std::expintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > std::sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`

- float [std::sph_legendre](#) (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::fp_promote_t<_Tp> [std::sph_neumann](#) (unsigned int __n, _Tp __x)
- float [std::sph_neumannf](#) (unsigned int __n, float __x)
- long double [std::sph_neumannl](#) (unsigned int __n, long double __x)

8.2.1 Detailed Description

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

8.2.2 Function Documentation

8.2.2.1 [assoc_laguerre\(\)](#)

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 degree n , nonnegative order m and real argument x .

The associated Laguerre function of real order 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 order $\alpha = m$ 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

<code>__n</code>	The degree of the Laguerre function, <code>__n >= 0</code> .
<code>__m</code>	The order of the Laguerre function, <code>__m >= 0</code> .
<code>__x</code>	The argument of the Laguerre function, <code>__x >= 0</code> .

Exceptions

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

Definition at line 422 of file `specfun.h`.

8.2.2.2 `assoc_laguerref()`

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

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

See also

[assoc_laguerre](#) for more details.

Definition at line 374 of file `specfun.h`.

8.2.2.3 `assoc_laguerrel()`

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

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

See also

[assoc_laguerre](#) for more details.

Definition at line 385 of file `specfun.h`.

8.2.2.4 `assoc_legendre()`

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

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

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

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

See also

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

Note

$$P_l^m(x) = 0 \text{ if } m > l.$$

Template Parameters

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

Parameters

<code>__l</code>	The degree <code>__l</code> ≥ 0 .
<code>__m</code>	The order <code>__m</code> .
<code>__x</code>	The argument, <code>abs (__x)</code> ≤ 1 .

Exceptions

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

Definition at line 471 of file `specfun.h`.

8.2.2.5 `assoc_legendref()`

```
float std::assoc_legendref (
    unsigned int __l,
```

```
unsigned int __m,
float __x ) [inline]
```

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

See also

[assoc_legendre](#) for more details.

Definition at line 437 of file specfun.h.

8.2.2.6 assoc_legendrel()

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

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

See also

[assoc_legendre](#) for more details.

Definition at line 448 of file specfun.h.

8.2.2.7 beta()

```
template<typename _Tpa , typename _Tpb >
__gnu_cxx::fp_promote_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

\leftrightarrow <code>__a</code>	The first argument of the beta function, <code>__a > 0</code> .
\leftrightarrow <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 516 of file `specfun.h`.

8.2.2.8 `betaf()`

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

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

See also

[beta](#) for more details.

Definition at line 485 of file `specfun.h`.

8.2.2.9 `betal()`

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

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

See also

[beta](#) for more details.

Definition at line 495 of file `specfun.h`.

8.2.2.10 `comp_ellint_1()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 564 of file `specfun.h`.

8.2.2.11 `comp_ellint_1f()`

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

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

See also

[comp_ellint_1](#) for details.

Definition at line 531 of file `specfun.h`.

8.2.2.12 `comp_ellint_1l()`

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

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

See also

[comp_ellint_1](#) for details.

Definition at line 541 of file `specfun.h`.

8.2.2.13 `comp_ellint_2()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 611 of file specfun.h.

8.2.2.14 `comp_ellint_2f()`

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

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

See also

[comp_ellint_2](#) for details.

Definition at line 579 of file specfun.h.

8.2.2.15 `comp_ellint_2l()`

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

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

See also

[comp_ellint_2](#) for details.

Definition at line 589 of file specfun.h.

8.2.2.16 `comp_ellint_3()`

```
template<typename _Tp , typename _Tpn >
__gnu_cxx::fp_promote_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 662 of file `specfun.h`.

8.2.2.17 `comp_ellint_3f()`

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

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

See also

[comp_ellint_3](#) for details.

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

8.2.2.18 `comp_ellint_3l()`

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

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

See also

[comp_ellint_3](#) for details.

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

8.2.2.19 `cyl_bessel_i()`

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::fp_promote_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</code> ≥ 0

Exceptions

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

Definition at line 708 of file `specfun.h`.

8.2.2.20 `cyl_bessel_if()`

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

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

See also

[cyl_bessel_i](#) for setails.

Definition at line 677 of file `specfun.h`.

8.2.2.21 `cyl_bessel_il()`

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

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

See also

[cyl_bessel_i](#) for setails.

Definition at line 687 of file specfun.h.

8.2.2.22 `cyl_bessel_j()`

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::fp_promote_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</code> ≥ 0

Exceptions

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

Definition at line 754 of file specfun.h.

8.2.2.23 `cyl_bessel_jf()`

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

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

See also

[cyl_bessel_j](#) for setails.

Definition at line 723 of file `specfun.h`.

8.2.2.24 `cyl_bessel_jl()`

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

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

See also

[cyl_bessel_j](#) for setails.

Definition at line 733 of file `specfun.h`.

8.2.2.25 `cyl_bessel_k()`

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::fp_promote_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 806 of file `specfun.h`.

8.2.2.26 `cyl_bessel_kf()`

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

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

See also

[cyl_bessel_k](#) for setails.

Definition at line 769 of file `specfun.h`.

8.2.2.27 `cyl_bessel_kl()`

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

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

See also

[cyl_bessel_k](#) for setails.

Definition at line 779 of file `specfun.h`.

8.2.2.28 `cyl_neumann()`

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::fp_promote_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</code> ≥ 0

Exceptions

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

Definition at line 854 of file `specfun.h`.

8.2.2.29 `cyl_neumannf()`

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

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

See also

[cyl_neumann](#) for setails.

Definition at line 821 of file `specfun.h`.

8.2.2.30 cyl_neumannl()

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

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

See also

[cyl_neumann](#) for setails.

Definition at line 831 of file specfun.h.

8.2.2.31 ellint_1()

```
template<typename _Tp , typename _Tpp >
__gnu_cxx::fp_promote_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 902 of file `specfun.h`.

8.2.2.32 `ellint_1f()`

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

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

See also

[ellint_1](#) for details.

Definition at line 869 of file `specfun.h`.

8.2.2.33 `ellint_1l()`

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

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

See also

[ellint_1](#) for details.

Definition at line 879 of file `specfun.h`.

8.2.2.34 `ellint_2()`

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

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

The incomplete elliptic integral of the second kind is defined as

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

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

See also

[comp_ellint_2](#).

Template Parameters

<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 950 of file `specfun.h`.

8.2.2.35 `ellint_2f()`

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

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

See also

[ellint_2](#) for details.

Definition at line 917 of file `specfun.h`.

8.2.2.36 `ellint_2l()`

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

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

See also

[ellint_2](#) for details.

Definition at line 927 of file `specfun.h`.

8.2.2.37 `ellint_3()`

```
template<typename _Tp , typename _Tpn , typename _Tpp >
__gnu_cxx::fp_promote_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 1003 of file `specfun.h`.

8.2.2.38 ellint_3f()

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

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

See also

[ellint_3](#) for details.

Definition at line 965 of file `specfun.h`.

8.2.2.39 ellint_3l()

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

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

See also

[ellint_3](#) for details.

Definition at line 975 of file `specfun.h`.

8.2.2.40 expint()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 1043 of file `specfun.h`.

8.2.2.41 `expintf()`

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

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

See also

[expint](#) for details.

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

8.2.2.42 `expintl()`

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

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

See also

[expint](#) for details.

Definition at line 1027 of file `specfun.h`.

8.2.2.43 hermite()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 1091 of file `specfun.h`.

8.2.2.44 hermitef()

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

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

See also

[hermite](#) for details.

Definition at line 1058 of file `specfun.h`.

8.2.2.45 `hermitel()`

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

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

See also

[hermite](#) for details.

Definition at line 1068 of file `specfun.h`.

8.2.2.46 `laguerre()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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</code> ≥ 0

Exceptions

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

Definition at line 1135 of file `specfun.h`.

8.2.2.47 laguerref()

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

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

See also

[laguerre](#) for more details.

Definition at line 1106 of file `specfun.h`.

8.2.2.48 laguerrel()

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

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

See also

[laguerre](#) for more details.

Definition at line 1116 of file `specfun.h`.

8.2.2.49 legendre()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 1180 of file `specfun.h`.

8.2.2.50 `legendref()`

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

Return the Legendre polynomial $P_l(x)$ of nonnegative degree l and `float` argument $|x| \leq 1$.

See also

[legendre](#) for more details.

Definition at line 1150 of file `specfun.h`.

8.2.2.51 `legendrel()`

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

Return the Legendre polynomial $P_l(x)$ of nonnegative degree l and `long double` argument $|x| \leq 1$.

See also

[legendre](#) for more details.

Definition at line 1160 of file `specfun.h`.

8.2.2.52 `riemann_zeta()`

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

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

The Riemann zeta function is defined by:

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

and

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

For $s < 1$ use the reflection formula:

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

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 1231 of file `specfun.h`.

8.2.2.53 `riemann_zetaf()`

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

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

See also

[riemann_zeta](#) for more details.

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

8.2.2.54 `riemann_zetal()`

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

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

See also

[riemann_zeta](#) for more details.

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

8.2.2.55 `sph_bessel()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 $n \geq 0$
<code>__x</code>	The real argument $x \geq 0$

Exceptions

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

Definition at line 1275 of file `specfun.h`.

8.2.2.56 sph_besself()

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

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

See also

[sph_bessel](#) for more details.

Definition at line 1246 of file `specfun.h`.

8.2.2.57 sph_bessell()

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

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

See also

[sph_bessel](#) for more details.

Definition at line 1256 of file `specfun.h`.

8.2.2.58 sph_legendre()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 1322 of file `specfun.h`.

8.2.2.59 `sph_legendref()`

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

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

See also

[sph_legendre](#) for details.

Definition at line 1290 of file `specfun.h`.

8.2.2.60 `sph_legendrel()`

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

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

See also

[sph_legendre](#) for details.

Definition at line 1301 of file `specfun.h`.

8.2.2.61 sph_neumann()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 1366 of file specfun.h.

8.2.2.62 sph_neumannf()

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

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

See also

[sph_neumann](#) for details.

Definition at line 1337 of file specfun.h.

8.2.2.63 sph_neumannl()

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

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

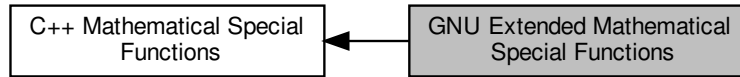
See also

[sph_neumann](#) for details.

Definition at line 1347 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::fp_promote_t< _Tp > __gnu_cxx::airy_ai (_Tp __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::airy_bi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::binomial_p (_Tp __p, unsigned int __n, unsigned int __k)`
Return the binomial cumulative distribution function.
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t<_Tp>](#) [__gnu_cxx::clausen](#) (unsigned int __m, _Tp __x)
- template<typename _Tp >
[std::complex<__gnu_cxx::fp_promote_t<_Tp>>](#) [__gnu_cxx::clausen](#) (unsigned int __m, [std::complex<_Tp>](#) __z)
- template<typename _Tp >
[__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t<_Tx, _Ty>](#) [__gnu_cxx::comp_ellint_rg](#) (_Tx __x, _Ty __y)

- `template<typename _Tpa, typename _Tpc, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tpa, _Tpc, _Tp > __gnu_cxx::conf_hyperg (_Tpa __a, _Tpc __c, _Tp __x)`
- `template<typename _Tpc, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tpc, _Tp > __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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::digamma (_Tp __x)`
- `float __gnu_cxx::digammaf (float __x)`
- `long double __gnu_cxx::digammal (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > __gnu_cxx::dilog (_Tp __x)`
- `float __gnu_cxx::dilogf (float __x)`
- `long double __gnu_cxx::dilogl (long double __x)`
- `template<typename _Tp >`
`_Tp __gnu_cxx::dirichlet_beta (_Tp __s)`
- `float __gnu_cxx::dirichlet_betaf (float __s)`
- `long double __gnu_cxx::dirichlet_betall (long double __s)`
- `template<typename _Tp >`
`_Tp __gnu_cxx::dirichlet_eta (_Tp __s)`
- `float __gnu_cxx::dirichlet_etaf (float __s)`
- `long double __gnu_cxx::dirichlet_etall (long double __s)`
- `template<typename _Tp >`
`_Tp __gnu_cxx::dirichlet_lambda (_Tp __s)`
- `float __gnu_cxx::dirichlet_lambdaf (float __s)`
- `long double __gnu_cxx::dirichlet_lambdall (long double __s)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t< _Tk, _Tphi > __gnu_cxx::ellint_d (_Tk __k, _Tphi __phi)`
- `float __gnu_cxx::ellint_d (float __k, float __phi)`
- `long double __gnu_cxx::ellint_dl (long double __k, long double __phi)`
- `template<typename _Tp, typename _Tk >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t](#)< _Tlam, _Tp > [__gnu_cxx::exponential_p](#) (_Tlam __lambda, _Tp __x)
Return the exponential cumulative probability density function.
 - template<typename _Tlam, typename _Tp >
[__gnu_cxx::fp_promote_t](#)< _Tlam, _Tp > [__gnu_cxx::exponential_pdf](#) (_Tlam __lambda, _Tp __x)
Return the exponential probability density function.
 - template<typename _Tp >
[__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t<_Tp>](#) [__gnu_cxx::fisher_f_p](#) (_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::fp_promote_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::fp_promote_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::fp_promote_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 _Tp >
[__gnu_cxx::fp_promote_t<_Ta, _Tp>](#) [__gnu_cxx::gamma_p](#) (_Ta __a, _Tp __x)
Return the gamma cumulative propability distribution function or the regularized lower incomplete gamma function.
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::fp_promote_t<_Ta, _Tb, _Tp>](#) [__gnu_cxx::gamma_pdf](#) (_Ta __alpha, _Tb __beta, _Tp __x)
Return the gamma propability distribution function.
- float [__gnu_cxx::gamma_pf](#) (float __a, float __x)
- long double [__gnu_cxx::gamma_pl](#) (long double __a, long double __x)
- template<typename _Ta, typename _Tp >
[__gnu_cxx::fp_promote_t<_Ta, _Tp>](#) [__gnu_cxx::gamma_q](#) (_Ta __a, _Tp __x)
Return the gamma complementary cumulative propability distribution (or survival) function or the regularized upper incomplete gamma function.
- float [__gnu_cxx::gamma_qf](#) (float __a, float __x)
- long double [__gnu_cxx::gamma_ql](#) (long double __a, long double __x)
- template<typename _Ta >
[__gnu_cxx::fp_promote_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 _Tlam, typename _Tp >
[__gnu_cxx::fp_promote_t<_Tlam, _Tp>](#) [__gnu_cxx::gegenbauer](#) (unsigned int __n, _Tlam __lambda, _Tp __x)
- float [__gnu_cxx::gegenbauerf](#) (unsigned int __n, float __lambda, float __x)
- long double [__gnu_cxx::gegenbauerl](#) (unsigned int __n, long double __lambda, long double __x)

- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > __gnu_cxx::harmonic` (unsigned int __n)
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta` (_Ta __a, _Tb __b, _Tp __x)
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Ta, _Tb, _Tp > __gnu_cxx::ibetac` (_Ta __a, _Tb __b, _Tp __x)
- `float __gnu_cxx::ibetaf` (float __a, float __b, float __x)
- `long double __gnu_cxx::ibetac1` (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::fp_promote_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi` (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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 _Tpq, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tpq, _Tp > __gnu_cxx::jacobi_theta_1` (_Tpq __q, _Tp __x)
- `float __gnu_cxx::jacobi_theta_1f` (float __q, float __x)
- `long double __gnu_cxx::jacobi_theta_1l` (long double __q, long double __x)
- `template<typename _Tpq, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tpq, _Tp > __gnu_cxx::jacobi_theta_2` (_Tpq __q, _Tp __x)
- `float __gnu_cxx::jacobi_theta_2f` (float __q, float __x)
- `long double __gnu_cxx::jacobi_theta_2l` (long double __q, long double __x)
- `template<typename _Tpq, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tpq, _Tp > __gnu_cxx::jacobi_theta_3` (_Tpq __q, _Tp __x)
- `float __gnu_cxx::jacobi_theta_3f` (float __q, float __x)
- `long double __gnu_cxx::jacobi_theta_3l` (long double __q, long double __x)

- `template<typename _Tpq, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tpq, _Tp > __gnu_cxx::jacobi_theta_4 (_Tpq __q, _Tp __x)`
- `float __gnu_cxx::jacobi_theta_4f (float __q, float __x)`
- `long double __gnu_cxx::jacobi_theta_4l (long double __q, long double __x)`
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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, typename _Ts, typename _Ta >`
`__gnu_cxx::fp_promote_t< _Tp, _Ts, _Ta > __gnu_cxx::lerch_phi (_Tp __z, _Ts __s, _Ta __a)`
- `float __gnu_cxx::lerch_phif (float __z, float __s, float __a)`
- `long double __gnu_cxx::lerch_phil (long double __z, long double __s, long double __a)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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

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

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.

- 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::fp_promote_t< _Ta > __gnu_cxx::lgamma](#) (_Ta __a)
- template<typename _Ta >
 std::complex< [__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t< _Ta, _Tb, _Tp > __gnu_cxx::logistic_p](#) (_Ta __a, _Tb __b, _Tp __x)

Return the logistic cumulative distribution function.

- template<typename _Ta, typename _Tb, typename _Tp >
 [__gnu_cxx::fp_promote_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::fp_promote_t< _Tmu, _Tsig, _Tp > __gnu_cxx::lognormal_p](#) (_Tmu __mu, _Tsig __sigma, _Tp __x)

Return the lognormal cumulative probability density function.

- template<typename _Tmu, typename _Tsig, typename _Tp >
 [__gnu_cxx::fp_promote_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::fp_promote_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

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

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::fp_promote_t<_Tmu, _Tsig, _Tp>](#) [__gnu_cxx::normal_p](#) (_Tmu __mu, _Tsig __sigma, _Tp __x)
Return the normal cumulative probability density function.
- template<typename _Tmu, typename _Tsig, typename _Tp >
[__gnu_cxx::fp_promote_t<_Tmu, _Tsig, _Tp>](#) [__gnu_cxx::normal_pdf](#) (_Tmu __mu, _Tsig __sigma, _Tp __x)
Return the gamma cumulative propability distribution function.
- template<typename _Tph, typename _Tpa >
[__gnu_cxx::fp_promote_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 _Tp >
[__gnu_cxx::fp_promote_t<_Tp>](#) [__gnu_cxx::polygamma](#) (unsigned int __m, _Tp __x)
- float [__gnu_cxx::polygammaf](#) (unsigned int __m, float __x)
- long double [__gnu_cxx::polygammal](#) (unsigned int __m, long double __x)
- template<typename _Tp, typename _Wp >
[__gnu_cxx::fp_promote_t<_Tp, _Wp>](#) [__gnu_cxx::polylog](#) (_Tp __s, _Wp __w)
- template<typename _Tp, typename _Wp >
[std::complex<__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t<_Tp>](#) [__gnu_cxx::sinc](#) (_Tp __x)
- template<typename _Tp >
[__gnu_cxx::fp_promote_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::fp_promote_t< _Tp > > [__gnu_cxx::sincos](#) (_Tp __x)
- template<typename _Tp >
 [__gnu_cxx::__sincos_t](#)< __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > [__gnu_cxx::sinhc](#) (_Tp __x)
- template<typename _Tp >
 __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > > [__gnu_cxx::sph_hankel_1](#) (unsigned int __n, _Tp __z)
- template<typename _Tp >
 std::complex< __gnu_cxx::fp_promote_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::fp_promote_t< _Tp > > [__gnu_cxx::sph_hankel_2](#) (unsigned int __n, _Tp __z)

- `template<typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::student_t_p (_Tt __t, unsigned int __nu)`
Return the Students T probability function.
- `template<typename _Tt, typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Ta > __gnu_cxx::tgamma (_Ta __a)`
- `template<typename _Ta >`
`std::complex< __gnu_cxx::fp_promote_t< _Ta > > __gnu_cxx::tgamma (std::complex< _Ta > __a)`
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Ta, _Tp > __gnu_cxx::tgamma (_Ta __a, _Tp __x)`
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Ta, _Tb, _Tp > __gnu_cxx::weibull_p (_Ta __a, _Tb __b, _Tp __x)`
Return the Weibull cumulative probability density function.
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_t< _Trho, _Tphi > __gnu_cxx::zernike (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)`
- `float __gnu_cxx::zernikef (unsigned int __n, int __m, float __rho, float __phi)`
- `long double __gnu_cxx::zernikel (unsigned int __n, int __m, long double __rho, long double __phi)`

8.3.1 Detailed Description

An extended collection of advanced mathematical special functions for GNU.

8.3.2 Function Documentation

8.3.2.1 `airy_ai()` [1/2]

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

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

The Airy function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 2821 of file `specfun.h`.

8.3.2.2 `airy_ai()` [2/2]

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

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

The Airy function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 2841 of file specfun.h.

8.3.2.3 `airy_aif()`

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

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

See also

[airy_ai](#) for details.

Definition at line 2794 of file specfun.h.

8.3.2.4 `airy_ail()`

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

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

See also

[airy_ai](#) for details.

Definition at line 2804 of file specfun.h.

8.3.2.5 `airy_bi()` [1/2]

```
template<typename _Tp >
__gnu_cxx::fp_promote_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>_↔</code> <code>_x</code>	The argument
------------------------------------	--------------

Definition at line 2883 of file `specfun.h`.

8.3.2.6 `airy_bi()` [2/2]

```
template<typename _Tp >
std::complex<__gnu_cxx::fp_promote_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>_↔</code> <code>_x</code>	The complex argument
------------------------------------	----------------------

Definition at line 2904 of file `specfun.h`.

8.3.2.7 `airy_bif()`

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

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

See also

[airy_bi](#) for details.

Definition at line 2855 of file specfun.h.

8.3.2.8 airy_bil()

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

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

See also

[airy_bi](#) for details.

Definition at line 2865 of file specfun.h.

8.3.2.9 bernoulli() [1/2]

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

\leftrightarrow	The order.
n	

Definition at line 4318 of file specfun.h.

8.3.2.10 `bernoulli()` [2/2]

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

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

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

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

The derivative is proportional to the previous polynomial:

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

The series expansion for the Bernoulli polynomials is:

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

A useful argument promotion is:

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

Definition at line 6880 of file `specfun.h`.

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

8.3.2.11 `bernoullicf()`

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

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

See also

[bernoulli](#) for details.

Definition at line 4291 of file `specfun.h`.

8.3.2.12 bernoullil()

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

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

See also

[bernoulli](#) for details.

Definition at line 4301 of file specfun.h.

8.3.2.13 binomial()

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

\hookleftarrow <code>__n</code>	The first argument of the binomial coefficient.
\hookleftarrow <code>__k</code>	The second argument of the binomial coefficient.

Returns

The binomial coefficient.

Definition at line 4234 of file specfun.h.

8.3.2.14 `binomial_p()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_t<_Tp> __gnu_cxx::binomial_p (
    _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 6733 of file specfun.h.

8.3.2.15 `binomial_pdf()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 6712 of file specfun.h.

8.3.2.16 binomialf()

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

Return the binomial coefficient as a `float`.

See also

[binomial](#) for details.

Definition at line 4205 of file specfun.h.

8.3.2.17 binomiall()

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

Return the binomial coefficient as a `long double`.

See also

[binomial](#) for details.

Definition at line 4214 of file specfun.h.

8.3.2.18 bose_einstein()

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

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

See also

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

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

Parameters

$_s$	The order $s \geq 0$.
$_x$	The real argument.

Returns

The real Bose-Einstein integral $G_s(x)$,

Definition at line 6111 of file specfun.h.

8.3.2.19 `bose_einseinf()`

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

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

See also

[bose_einstein](#) for details.

Definition at line 6081 of file specfun.h.

8.3.2.20 `bose_einsteinl()`

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

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

See also

[bose_einstein](#) for details.

Definition at line 6091 of file specfun.h.

8.3.2.21 chebyshev_t()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 2052 of file specfun.h.

8.3.2.22 `chebyshev_tf()`

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

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

See also

[chebyshev_t](#) for details.

Definition at line 2023 of file specfun.h.

8.3.2.23 `chebyshev_tl()`

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

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

See also

[chebyshev_t](#) for details.

Definition at line 2033 of file specfun.h.

8.3.2.24 `chebyshev_u()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 2096 of file `specfun.h`.

8.3.2.25 `chebyshev_uf()`

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

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

See also

[chebyshev_u](#) for details.

Definition at line 2067 of file `specfun.h`.

8.3.2.26 `chebyshev_ul()`

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

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

See also

[chebyshev_u](#) for details.

Definition at line 2077 of file `specfun.h`.

8.3.2.27 `chebyshev_v()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 2141 of file `specfun.h`.

8.3.2.28 `chebyshev_vf()`

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

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

See also

[chebyshev_v](#) for details.

Definition at line 2111 of file `specfun.h`.

8.3.2.29 `chebyshev_vl()`

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

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

See also

[chebyshev_v](#) for details.

Definition at line 2121 of file `specfun.h`.

8.3.2.30 `chebyshev_w()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 2186 of file `specfun.h`.

8.3.2.31 `chebyshev_wf()`

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

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

See also

[chebyshev_w](#) for details.

Definition at line 2156 of file `specfun.h`.

8.3.2.32 `chebyshev_wl()`

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

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

See also

[chebyshev_w](#) for details.

Definition at line 2166 of file `specfun.h`.

8.3.2.33 `clausen()` [1/2]

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

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

The Clausen function is defined by

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

Template Parameters

<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 5362 of file `specfun.h`.

8.3.2.34 `clausen()` [2/2]

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

\leftrightarrow _m	The integral order
\leftrightarrow _z	The complex argument

Definition at line 5406 of file specfun.h.

8.3.2.35 `clausen_cl()`

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

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

Parameters

\leftrightarrow _m	The unsigned integer order
\leftrightarrow _x	The real argument

Definition at line 5317 of file specfun.h.

8.3.2.36 `clausen_clf()`

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

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

See also

[clausen_cl](#) for details.

Definition at line 5289 of file specfun.h.

8.3.2.37 clausen_cll()

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

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

See also

[clausen_cl](#) for details.

Definition at line 5299 of file specfun.h.

8.3.2.38 clausen_sl()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 5274 of file specfun.h.

8.3.2.39 `clausen_slf()`

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

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

See also

[clausen_sl](#) for details.

Definition at line 5246 of file specfun.h.

8.3.2.40 `clausen_sll()`

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

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

See also

[clausen_sl](#) for details.

Definition at line 5256 of file specfun.h.

8.3.2.41 `clausenf()` [1/2]

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

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

See also

[clausen](#) for details.

Definition at line 5332 of file specfun.h.

8.3.2.42 `clausenf()` [2/2]

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

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

See also

[clausen](#) for details.

Definition at line 5377 of file `specfun.h`.

8.3.2.43 `clausenl()` [1/2]

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

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

See also

[clausen](#) for details.

Definition at line 5342 of file `specfun.h`.

8.3.2.44 `clausenl()` [2/2]

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

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

See also

[clausen](#) for details.

Definition at line 5387 of file `specfun.h`.

8.3.2.45 `comp_ellint_d()`

```
template<typename _Tk >
__gnu_cxx::fp_promote_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

<code>__k</code>	The modulus $-1 \leq k \leq +1$.
------------------	-----------------------------------

Definition at line 4534 of file `specfun.h`.

8.3.2.46 `comp_ellint_df()`

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

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

See also

[comp_ellint_d](#) for details.

Definition at line 4507 of file `specfun.h`.

8.3.2.47 `comp_ellint_dl()`

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

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

See also

[comp_ellint_d](#) for details.

Definition at line 4517 of file `specfun.h`.

8.3.2.48 comp_ellint_rf() [1/3]

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

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

See also

[comp_ellint_rf](#) for details.

Definition at line 3164 of file specfun.h.

8.3.2.49 comp_ellint_rf() [2/3]

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

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

See also

[comp_ellint_rf](#) for details.

Definition at line 3174 of file specfun.h.

8.3.2.50 comp_ellint_rf() [3/3]

```
template<typename _Tx , typename _Ty >
__gnu_cxx::fp_promote_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 3192 of file specfun.h.

8.3.2.51 `comp_ellint_rg()` [1/3]

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

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

See also

[comp_ellint_rg](#) for details.

Definition at line 3397 of file specfun.h.

8.3.2.52 `comp_ellint_rg()` [2/3]

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

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

See also

[comp_ellint_rg](#) for details.

Definition at line 3406 of file specfun.h.

8.3.2.53 `comp_ellint_rg()` [3/3]

```
template<typename _Tx , typename _Ty >
__gnu_cxx::fp_promote_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 3425 of file specfun.h.

8.3.2.54 conf_hyperg()

```
template<typename _Tpa , typename _Tpc , typename _Tp >
__gnu_cxx::fp_promote_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 numerator parameter a , denominator 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 numerator parameter
\leftrightarrow _c	The denominator parameter
\leftrightarrow _x	The argument

Definition at line 1431 of file specfun.h.

8.3.2.55 conf_hyperg_lim()

```
template<typename _Tpc , typename _Tp >
__gnu_cxx::fp_promote_t<_Tpc, _Tp> __gnu_cxx::conf_hyperg_lim (
```

```

_Tpc __c,
_Tp __x ) [inline]

```

Return the confluent hypergeometric limit function ${}_0F_1(; c; x)$ of real numerator 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

<code>__c</code>	The denominator parameter
<code>__x</code>	The argument

Definition at line 1576 of file specfun.h.

8.3.2.56 `conf_hyperg_limf()`

```

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

```

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

See also

[conf_hyperg_lim](#) for details.

Definition at line 1547 of file specfun.h.

8.3.2.57 `conf_hyperg_liml()`

```

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

```

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

See also

[conf_hyperg_lim](#) for details.

Definition at line 1557 of file specfun.h.

8.3.2.58 `conf_hypergf()`

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

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

See also

[conf_hyperg](#) for details.

Definition at line 1399 of file `specfun.h`.

8.3.2.59 `conf_hypergl()`

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

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

See also

[conf_hyperg](#) for details.

Definition at line 1410 of file `specfun.h`.

8.3.2.60 `cos_pi()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 6237 of file specfun.h.

8.3.2.61 `cos_pif()`

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

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

See also

[cos_pi](#) for more details.

Definition at line 6210 of file specfun.h.

8.3.2.62 `cos_pil()`

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

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

See also

[cos_pi](#) for more details.

Definition at line 6220 of file specfun.h.

8.3.2.63 `cosh_pi()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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>__x</code>	The argument
------------------	--------------

Definition at line 6279 of file `specfun.h`.

8.3.2.64 `cosh_pif()`

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

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

See also

[cosh_pi](#) for more details.

Definition at line 6252 of file `specfun.h`.

8.3.2.65 `cosh_pil()`

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

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

See also

[cosh_pi](#) for more details.

Definition at line 6262 of file `specfun.h`.

8.3.2.66 `coshint()`

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

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

The hyperbolic cosine integral is defined by

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

Template Parameters

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

Parameters

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

Definition at line 1858 of file `specfun.h`.

8.3.2.67 `coshintf()`

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

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

See also

[coshint](#) for details.

Definition at line 1830 of file `specfun.h`.

8.3.2.68 `coshintl()`

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

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

See also

[coshint](#) for details.

Definition at line 1840 of file `specfun.h`.

8.3.2.69 `cosint()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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>__x</code>	The real upper integration limit
------------------	----------------------------------

Definition at line 1775 of file specfun.h.

8.3.2.70 `cosintf()`

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

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

See also

[cosint](#) for details.

Definition at line 1749 of file specfun.h.

8.3.2.71 `cosintl()`

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

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

See also

[cosint](#) for details.

Definition at line 1759 of file specfun.h.

8.3.2.72 `cyl_hankel_1()` [1/2]

```
template<typename _Tpnu , typename _Tp >
std::complex<__gnu_cxx::fp_promote_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 2548 of file `specfun.h`.

8.3.2.73 `cyl_hankel_1()` [2/2]

```
template<typename _Tpnu , typename _Tp >
std::complex<__gnu_cxx::fp_promote_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 4811 of file `specfun.h`.

8.3.2.74 `cyl_hankel_1f()` [1/2]

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

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

See also

[cyl_hankel_1](#) for details.

Definition at line 2516 of file specfun.h.

8.3.2.75 `cyl_hankel_1f()` [2/2]

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

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

See also

[cyl_hankel_1](#) for more details.

Definition at line 4780 of file specfun.h.

8.3.2.76 `cyl_hankel_1l()` [1/2]

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

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

See also

[cyl_hankel_1](#) for details.

Definition at line 2527 of file specfun.h.

8.3.2.77 `cyl_hankel_1l()` [2/2]

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

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

See also

[cyl_hankel_1](#) for more details.

Definition at line 4791 of file `specfun.h`.

8.3.2.78 `cyl_hankel_2()` [1/2]

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

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

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

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

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 2596 of file `specfun.h`.

8.3.2.79 `cyl_hankel_2()` [2/2]

```
template<typename _Tpnu , typename _Tp >
std::complex<__gnu_cxx::fp_promote_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 4858 of file `specfun.h`.

8.3.2.80 `cyl_hankel_2f()` [1/2]

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

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

See also

[cyl_hankel_2](#) for details.

Definition at line 2564 of file `specfun.h`.

8.3.2.81 `cyl_hankel_2f()` [2/2]

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

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

See also

[cyl_hankel_2](#) for more details.

Definition at line 4827 of file specfun.h.

8.3.2.82 `cyl_hankel_2l()` [1/2]

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

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

See also

[cyl_hankel_2](#) for details.

Definition at line 2575 of file specfun.h.

8.3.2.83 `cyl_hankel_2l()` [2/2]

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

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

See also

[cyl_hankel_2](#) for more details.

Definition at line 4838 of file specfun.h.

8.3.2.84 dawson()

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

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

The Dawson integral is defined by:

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

and it's derivative is:

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

Parameters

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

Definition at line 3808 of file specfun.h.

8.3.2.85 dawsonf()

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

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

See also

[dawson](#) for details.

Definition at line 3779 of file specfun.h.

8.3.2.86 dawsonl()

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

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

See also

[dawson](#) for details.

Definition at line 3789 of file specfun.h.

8.3.2.87 debye()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 6849 of file `specfun.h`.

8.3.2.88 debyef()

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

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

See also

[debye](#) for details.

Definition at line 6821 of file `specfun.h`.

8.3.2.89 debyel()

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

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

See also

[debye](#) for details.

Definition at line 6831 of file specfun.h.

8.3.2.90 digamma()

```
template<typename _Tp >
__gnu_cxx::fp_promote_t<_Tp> __gnu_cxx::digamma (
    _Tp __x ) [inline]
```

Return the digamma or psi function of argument x .

The the digamma or psi function is defined by

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

the logarithmic derivative of the gamma function.

Parameters

\leftrightarrow	The parameter
x	

Definition at line 3571 of file specfun.h.

8.3.2.91 digammaf()

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

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

See also

[digamma](#) for details.

Definition at line 3544 of file specfun.h.

8.3.2.92 digammal()

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

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

See also

[digamma](#) for details.

Definition at line 3554 of file specfun.h.

8.3.2.93 dilog()

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

Return the dilogarithm function $Li_2(z)$ 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 3149 of file specfun.h.

8.3.2.94 dlogf()

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

Return the dilogarithm function $Li_2(z)$ for `float` argument.

See also

[dilog](#) for details.

Definition at line 3123 of file `specfun.h`.

8.3.2.95 dilogl()

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

Return the dilogarithm function $Li_2(z)$ for `long double` argument.

See also

[dilog](#) for details.

Definition at line 3133 of file `specfun.h`.

8.3.2.96 dirichlet_beta()

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

Return the Dirichlet beta function of real argument s .

The Dirichlet beta function is defined by:

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

An important reflection formula is:

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

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

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

Parameters

$_s$	
-------	--

Definition at line 5188 of file specfun.h.

8.3.2.97 `dirichlet_betaf()`

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

Return the Dirichlet beta function of real argument s .

See also

[dirichlet_beta](#) for details.

Definition at line 5153 of file specfun.h.

8.3.2.98 `dirichlet_betal()`

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

Return the Dirichlet beta function of real argument s .

See also

[dirichlet_beta](#) for details.

Definition at line 5162 of file specfun.h.

8.3.2.99 `dirichlet_eta()`

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

Return the Dirichlet eta function of real argument s .

The Dirichlet eta function is defined by

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

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

\leftrightarrow	
s	

Definition at line 5139 of file specfun.h.

8.3.2.100 `dirichlet_etaf()`

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

Return the Dirichlet eta function of real argument s .

See also

[dirichlet_eta](#) for details.

Definition at line 5103 of file specfun.h.

8.3.2.101 `dirichlet_etal()`

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

Return the Dirichlet eta function of real argument s .

See also

[dirichlet_eta](#) for details.

Definition at line 5112 of file specfun.h.

8.3.2.102 `dirichlet_lambda()`

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

Return the Dirichlet lambda function of real argument s .

The Dirichlet lambda function is defined by

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

In terms of the Riemann zeta and the Dirichlet eta functions

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

Parameters

\leftrightarrow	
<code>__s</code>	

Definition at line 5231 of file specfun.h.

8.3.2.103 `dirichlet_lambdaf()`

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

Return the Dirichlet lambda function of real argument s .

See also

[dirichlet_lambda](#) for details.

Definition at line 5202 of file specfun.h.

8.3.2.104 `dirichlet_lambdal()`

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

Return the Dirichlet lambda function of real argument s .

See also

[dirichlet_lambda](#) for details.

Definition at line 5211 of file specfun.h.

8.3.2.105 `double_factorial()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 4112 of file specfun.h.

8.3.2.106 double_factorialf()

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

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

See also

[double_factorial](#) for more details

Definition at line 4085 of file `specfun.h`.

8.3.2.107 double_factoriall()

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

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

See also

[double_factorial](#) for more details

Definition at line 4095 of file `specfun.h`.

8.3.2.108 ellint_cel()

```
template<typename _Tk , typename _Tp , typename _Ta , typename _Tb >
__gnu_cxx::fp_promote_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 4764 of file specfun.h.

8.3.2.109 `ellint_celf()`

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

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

See also

[ellint_cel](#) for details.

Definition at line 4732 of file specfun.h.

8.3.2.110 `ellint_cell()`

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

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

See also

[ellint_cel](#) for details.

Definition at line 4741 of file specfun.h.

8.3.2.111 ellint_d()

```
template<typename _Tk , typename _Tphi >
__gnu_cxx::fp_promote_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 4577 of file specfun.h.

8.3.2.112 ellint_df()

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

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

See also

[ellint_d](#) for details.

Definition at line 4549 of file specfun.h.

8.3.2.113 ellint_dl()

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

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

See also

[ellint_d](#) for details.

Definition at line 4559 of file specfun.h.

8.3.2.114 `ellint_el1()`

```
template<typename _Tp , typename _Tk >
__gnu_cxx::fp_promote_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 4623 of file `specfun.h`.

8.3.2.115 `ellint_el1f()`

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

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

See also

[ellint_el1](#) for details.

Definition at line 4593 of file `specfun.h`.

8.3.2.116 `ellint_el1l()`

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

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

See also

[ellint_el1](#) for details.

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

8.3.2.117 `ellint_el2()`

```
template<typename _Tp , typename _Tk , typename _Ta , typename _Tb >
__gnu_cxx::fp_promote_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 4669 of file `specfun.h`.

8.3.2.118 `ellint_el2f()`

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

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

See also

[ellint_el2](#) for details.

Definition at line 4638 of file `specfun.h`.

8.3.2.119 `ellint_el2l()`

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

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

See also

[ellint_el2](#) for details.

Definition at line 4648 of file specfun.h.

8.3.2.120 `ellint_el3()`

```
template<typename _Tx , typename _Tk , typename _Tp >
__gnu_cxx::fp_promote_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 4716 of file specfun.h.

8.3.2.121 ellint_el3f()

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

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

See also

[ellint_el3](#) for details.

Definition at line 4685 of file `specfun.h`.

8.3.2.122 ellint_el3l()

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

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

See also

[ellint_el3](#) for details.

Definition at line 4696 of file `specfun.h`.

8.3.2.123 ellint_rc()

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

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

The Carlson elliptic function is defined by:

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 3284 of file specfun.h.

8.3.2.124 ellint_rcf()

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

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

See also

[ellint_rc](#) for details.

Definition at line 3250 of file specfun.h.

8.3.2.125 ellint_rcl()

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

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

See also

[ellint_rc](#) for details.

Definition at line 3259 of file specfun.h.

8.3.2.126 `ellint_rd()`

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

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

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

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 3383 of file `specfun.h`.

8.3.2.127 `ellint_rdf()`

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

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

See also

[ellint_rd](#) for details.

Definition at line 3347 of file `specfun.h`.

8.3.2.128 ellint_rdl()

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

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

See also

[ellint_rd](#) for details.

Definition at line 3356 of file specfun.h.

8.3.2.129 ellint_rf()

```
template<typename _Tp , typename _Up , typename _Vp >
__gnu_cxx::fp_promote_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

\leftrightarrow _x	The first of three symmetric arguments.
\leftrightarrow _y	The second of three symmetric arguments.
\leftrightarrow _z	The third of three symmetric arguments.

Definition at line 3236 of file specfun.h.

8.3.2.130 ellint_rff()

```
float __gnu_cxx::ellint_rff (
    float __x,
```

```
float __y,
float __z ) [inline]
```

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

See also

[ellint_rf](#) for details.

Definition at line 3207 of file `specfun.h`.

8.3.2.131 `ellint_rfl()`

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

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

See also

[ellint_rf](#) for details.

Definition at line 3217 of file `specfun.h`.

8.3.2.132 `ellint_rg()`

```
template<typename _Tp , typename _Up , typename _Vp >
__gnu_cxx::fp_promote_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

$_x$	The first of three symmetric arguments.
$_y$	The second of three symmetric arguments.
$_z$	The third of three symmetric arguments.

Definition at line 3474 of file specfun.h.

8.3.2.133 `ellint_rgf()`

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

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

See also

[ellint_rg](#) for details.

Definition at line 3439 of file specfun.h.

8.3.2.134 `ellint_rgl()`

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

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

See also

[ellint_rg](#) for details.

Definition at line 3448 of file specfun.h.

8.3.2.135 `ellint_rj()`

```
template<typename _Tp , typename _Up , typename _Vp , typename _Wp >
__gnu_cxx::fp_promote_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 3333 of file `specfun.h`.

8.3.2.136 `ellint_rjf()`

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

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

See also

[ellint_rj](#) for details.

Definition at line 3298 of file specfun.h.

8.3.2.137 `ellint_rjl()`

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

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

See also

[ellint_rj](#) for details.

Definition at line 3307 of file specfun.h.

8.3.2.138 `ellnome()`

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

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

The elliptic nome function is defined by

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

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

Template Parameters

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

Parameters

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

Definition at line 5620 of file specfun.h.

8.3.2.139 `ellnomef()`

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

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

See also

[ellnome](#) for details.

Definition at line 5593 of file specfun.h.

8.3.2.140 `ellnomel()`

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

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

See also

[ellnome](#) for details.

Definition at line 5603 of file specfun.h.

8.3.2.141 `euler()`

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

This returns Euler number E_n .

Parameters

<code>_↔</code>	the order n of the Euler number.
<code>_n</code>	

Returns

The Euler number of order n.

Definition at line 6891 of file specfun.h.

8.3.2.142 `eulerian_1()`

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

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

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

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

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

Definition at line 6909 of file specfun.h.

8.3.2.143 `eulerian_2()`

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

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

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

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

Definition at line 6927 of file specfun.h.

8.3.2.144 `expint()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 3854 of file `specfun.h`.

8.3.2.145 `expintf()`

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

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

See also

[expint](#) for details.

Definition at line 3823 of file `specfun.h`.

8.3.2.146 expintl()

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

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

See also

[expint](#) for details.

Definition at line 3833 of file specfun.h.

8.3.2.147 exponential_p()

```
template<typename _Tlam , typename _Tp >
__gnu_cxx::fp_promote_t<_Tlam, _Tp> __gnu_cxx::exponential_p (
    _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 6568 of file specfun.h.

8.3.2.148 exponential_pdf()

```
template<typename _Tlam , typename _Tp >
__gnu_cxx::fp_promote_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 6552 of file specfun.h.

8.3.2.149 factorial()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 4071 of file specfun.h.

8.3.2.150 factorialf()

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

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

See also

[factorial](#) for more details

Definition at line 4051 of file specfun.h.

8.3.2.151 factoriall()

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

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

See also

[factorial](#) for more details

Definition at line 4060 of file specfun.h.

8.3.2.152 falling_factorial()

```
template<typename _Tp, typename _Tnu >
__gnu_cxx::fp_promote_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

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

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

Definition at line 4037 of file specfun.h.

8.3.2.153 falling_factorialf()

```
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 4011 of file specfun.h.

8.3.2.154 falling_factoriall()

```
long double __gnu_cxx::falling_factoriall (
    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 4021 of file specfun.h.

8.3.2.155 `fermi_dirac()`

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

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

See also

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

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

Parameters

<code>__s</code>	The order <code>s > -1</code> .
<code>__x</code>	The real argument.

Returns

The real Fermi-Dirac integral `F_s(x)`,

Definition at line 6067 of file `specfun.h`.

8.3.2.156 `fermi_diracf()`

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

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

See also

[fermi_dirac](#) for details.

Definition at line 6037 of file `specfun.h`.

8.3.2.157 fermi_diracl()

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

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

See also

[fermi_dirac](#) for details.

Definition at line 6047 of file specfun.h.

8.3.2.158 fisher_f_p()

```
template<typename _Tp >
__gnu_cxx::fp_promote_t<_Tp> __gnu_cxx::fisher_f_p (
    _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 6666 of file specfun.h.

8.3.2.159 fisher_f_pdf()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 6691 of file specfun.h.

8.3.2.160 fresnel_c()

```

template<typename _Tp >
__gnu_cxx::fp_promote_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 3765 of file specfun.h.

8.3.2.161 fresnel_cf()

```

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

```

Definition at line 3746 of file specfun.h.

8.3.2.162 fresnel_cl()

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

Definition at line 3750 of file specfun.h.

8.3.2.163 fresnel_s()

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

$_x$	The argument
-------	--------------

Definition at line 3737 of file specfun.h.

8.3.2.164 fresnel_sf()

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

Definition at line 3718 of file specfun.h.

8.3.2.165 fresnel_sl()

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

Definition at line 3722 of file specfun.h.

8.3.2.166 gamma_p()

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

Return the gamma cumulative propability distribution function or the regularized lower incomplete gamma 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 4395 of file specfun.h.

8.3.2.167 gamma_pdf()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::fp_promote_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 6453 of file specfun.h.

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

8.3.2.168 gamma_pf()

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

Definition at line 4376 of file specfun.h.

8.3.2.169 gamma_pl()

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

Definition at line 4380 of file specfun.h.

8.3.2.170 gamma_q()

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

Return the gamma complementary cumulative propability distribution (or survival) function or the regularized upper incomplete gamma 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 4423 of file specfun.h.

8.3.2.171 gamma_qf()

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

Definition at line 4404 of file specfun.h.

8.3.2.172 `gamma_qf()`

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

Definition at line 4408 of file `specfun.h`.

8.3.2.173 `gamma_reciprocal()`

```
template<typename _Ta >
__gnu_cxx::fp_promote_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 6806 of file `specfun.h`.

8.3.2.174 `gamma_reciprocalf()`

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

Return the reciprocal gamma function for `float` argument.

See also

[gamma_reciprocal](#) for details.

Definition at line 6781 of file `specfun.h`.

8.3.2.175 gamma_reciprocall()

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

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

See also

[gamma_reciprocal](#) for details.

Definition at line 6791 of file `specfun.h`.

8.3.2.176 gegenbauer()

```
template<typename _Tlam , typename _Tp >
__gnu_cxx::fp_promote_t<_Tlam, _Tp> __gnu_cxx::gegenbauer (
    unsigned int __n,
    _Tlam __lambda,
    _Tp __x ) [inline]
```

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

The Gegenbauer polynomial is generated by a three-term recursion relation:

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

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

Template Parameters

<code>_Tlam</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>__lambda</code>	The real order
<code>__x</code>	The real argument

Definition at line 2307 of file `specfun.h`.

8.3.2.177 gegenbauerf()

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

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

See also

[gegenbauer](#) for details.

Definition at line 2270 of file specfun.h.

8.3.2.178 gegenbauerl()

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

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

See also

[gegenbauer](#) for details.

Definition at line 2281 of file specfun.h.

8.3.2.179 harmonic()

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

\leftrightarrow	The parameter
n	

Definition at line 3629 of file specfun.h.

8.3.2.180 `heuman_lambda()`

```
template<typename _Tk , typename _Tphi >
__gnu_cxx::fp_promote_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 4492 of file specfun.h.

8.3.2.181 `heuman_lambdaf()`

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

Definition at line 4466 of file specfun.h.

8.3.2.182 heuman_lambdal()

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

Definition at line 4470 of file specfun.h.

8.3.2.183 hurwitz_zeta() [1/2]

```
template<typename _Tp , typename _Up >
__gnu_cxx::fp_promote_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 3516 of file specfun.h.

8.3.2.184 hurwitz_zeta() [2/2]

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

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

See also

[hurwitz_zeta](#) for details.

Definition at line 3530 of file specfun.h.

8.3.2.185 hurwitz_zetaf()

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

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

See also

[hurwitz_zeta](#) for details.

Definition at line 3489 of file `specfun.h`.

8.3.2.186 hurwitz_zetal()

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

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

See also

[hurwitz_zeta](#) for details.

Definition at line 3499 of file `specfun.h`.

8.3.2.187 hyperg()

```
template<typename _Tpa , typename _Tpb , typename _Tpc , typename _Tp >
__gnu_cxx::fp_promote_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 numerator parameters a and b , denominator parameter c , and argument x .

The hypergeometric function is defined by

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

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

Parameters

\leftrightarrow _a	The first numerator parameter
\leftrightarrow _b	The second numerator parameter
\leftrightarrow _c	The denominator parameter
\leftrightarrow _x	The argument

Definition at line 1530 of file specfun.h.

8.3.2.188 hypergf()

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

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

See also

[hyperg](#) for details.

Definition at line 1497 of file specfun.h.

8.3.2.189 hypergl()

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

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

See also

[hyperg](#) for details.

Definition at line 1508 of file specfun.h.

8.3.2.190 ibeta()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::fp_promote_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

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

Definition at line 3678 of file specfun.h.

8.3.2.191 ibetac()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::fp_promote_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 3709 of file specfun.h.

8.3.2.192 ibetacf()

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

Definition at line 3687 of file specfun.h.

References `__gnu_cxx::ibetaf()`.

8.3.2.193 ibetacl()

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

Definition at line 3691 of file specfun.h.

References `__gnu_cxx::ibetal()`.

8.3.2.194 ibetaf()

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

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

See `ibeta` for details.

Definition at line 3644 of file specfun.h.

Referenced by `__gnu_cxx::ibetacf()`.

8.3.2.195 ibetal()

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

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

See `ibeta` for details.

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

Referenced by `__gnu_cxx::ibetacl()`.

8.3.2.196 jacobi()

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

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

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

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

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

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 2253 of file `specfun.h`.

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

8.3.2.197 `jacobi_cn()`

```
template<typename _Kp , typename _Up >
__gnu_cxx::fp_promote_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 1958 of file `specfun.h`.

8.3.2.198 `jacobi_cnf()`

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

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

See also

[jacobi_cn](#) for details.

Definition at line 1923 of file specfun.h.

8.3.2.199 `jacobi_cnl()`

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

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

See also

[jacobi_cn](#) for details.

Definition at line 1935 of file specfun.h.

8.3.2.200 `jacobi_dn()`

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

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

The Jacobi elliptic `dn` integral is defined by

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

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

See also

`ellint_1`).

Template Parameters

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

Parameters

\leftrightarrow _k	The real modulus
\leftrightarrow _u	The real argument

Definition at line 2008 of file specfun.h.

8.3.2.201 `jacobi_dnf()`

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

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

See also

[jacobi_dn](#) for details.

Definition at line 1973 of file specfun.h.

8.3.2.202 `jacobi_dnl()`

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

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

See also

[jacobi_dn](#) for details.

Definition at line 1985 of file specfun.h.

8.3.2.203 `jacobi_sn()`

```
template<typename _Kp , typename _Up >
__gnu_cxx::fp_promote_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 1908 of file `specfun.h`.

8.3.2.204 `jacobi_snf()`

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

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

See also

[jacobi_sn](#) for details.

Definition at line 1873 of file `specfun.h`.

8.3.2.205 `jacobi_sn()`

```
long double __gnu_cxx::jacobi_sn1 (
    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 1885 of file `specfun.h`.

8.3.2.206 `jacobi_theta_1()`

```
template<typename _Tpq , typename _Tp >
__gnu_cxx::fp_promote_t<_Tpq, _Tp> __gnu_cxx::jacobi_theta_1 (
    _Tpq __q,
    _Tp __x ) [inline]
```

Return the Jacobi theta-1 function $\theta_1(q, x)$ of nome q and argument x .

The Jacobi theta-1 function is defined by

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

Parameters

\leftrightarrow <code>_q</code>	The periodic (period = 2) argument
\leftrightarrow <code>_x</code>	The argument

Definition at line 5851 of file `specfun.h`.

8.3.2.207 `jacobi_theta_1f()`

```
float __gnu_cxx::jacobi_theta_1f (
    float __q,
    float __x ) [inline]
```

Return the Jacobi theta-1 function $\theta_1(q, x)$ of nome q and argument x .

See also

[jacobi_theta_1](#) for details.

Definition at line 5823 of file specfun.h.

8.3.2.208 `jacobi_theta_1l()`

```
long double __gnu_cxx::jacobi_theta_1l (
    long double __q,
    long double __x ) [inline]
```

Return the Jacobi theta-1 function $\theta_1(q, x)$ of nome q and argument x .

See also

[jacobi_theta_1](#) for details.

Definition at line 5833 of file specfun.h.

8.3.2.209 `jacobi_theta_2()`

```
template<typename _Tpq , typename _Tp >
__gnu_cxx::fp_promote_t<_Tpq, _Tp> __gnu_cxx::jacobi_theta_2 (
    _Tpq __q,
    _Tp __x ) [inline]
```

Return the Jacobi theta-2 function $\theta_2(q, x)$ of nome q and argument x .

The Jacobi theta-2 function is defined by

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

Parameters

\leftrightarrow <code>__q</code>	The periodic (period = 2) argument
\leftrightarrow <code>__x</code>	The argument

Definition at line 5894 of file specfun.h.

8.3.2.210 jacobi_theta_2f()

```
float __gnu_cxx::jacobi_theta_2f (
    float __q,
    float __x ) [inline]
```

Return the Jacobi theta-2 function $\theta_2(q, x)$ of nome q and argument x .

See also

[jacobi_theta_2](#) for details.

Definition at line 5866 of file specfun.h.

8.3.2.211 jacobi_theta_2l()

```
long double __gnu_cxx::jacobi_theta_2l (
    long double __q,
    long double __x ) [inline]
```

Return the Jacobi theta-2 function $\theta_2(q, x)$ of nome q and argument x .

See also

[jacobi_theta_2](#) for details.

Definition at line 5876 of file specfun.h.

8.3.2.212 jacobi_theta_3()

```
template<typename _Tpq , typename _Tp >
__gnu_cxx::fp_promote_t<_Tpq, _Tp> __gnu_cxx::jacobi_theta_3 (
    _Tpq __q,
    _Tp __x ) [inline]
```

Return the Jacobi theta-3 function $\theta_3(q, x)$ of nome q and argument x .

The Jacobi theta-3 function is defined by

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

Parameters

\leftrightarrow _q	The elliptic nome
\leftrightarrow _x	The argument

Definition at line 5937 of file specfun.h.

8.3.2.213 jacobi_theta_3f()

```
float __gnu_cxx::jacobi_theta_3f (
    float __q,
    float __x ) [inline]
```

Return the Jacobi theta-3 function $\theta_3(q, x)$ of nome q and argument x .

See also

[jacobi_theta_3](#) for details.

Definition at line 5909 of file specfun.h.

8.3.2.214 jacobi_theta_3l()

```
long double __gnu_cxx::jacobi_theta_3l (
    long double __q,
    long double __x ) [inline]
```

Return the Jacobi theta-3 function $\theta_3(q, x)$ of nome q and argument x .

See also

[jacobi_theta_3](#) for details.

Definition at line 5919 of file specfun.h.

8.3.2.215 jacobi_theta_4()

```
template<typename _Tpq , typename _Tp >
__gnu_cxx::fp_promote_t<_Tpq, _Tp> __gnu_cxx::jacobi_theta_4 (
    _Tpq __q,
    _Tp __x ) [inline]
```

Return the Jacobi theta-4 function $\theta_4(q, x)$ of nome q and argument x .

The Jacobi theta-4 function is defined by

$$\theta_4(q, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(q + j + 1/2)^2}{x}\right)$$

Parameters

\leftrightarrow _q	The elliptic nome
\leftrightarrow _x	The argument

Definition at line 5980 of file specfun.h.

8.3.2.216 jacobi_theta_4f()

```
float __gnu_cxx::jacobi_theta_4f (
    float __q,
    float __x ) [inline]
```

Return the Jacobi theta-4 function $\theta_4(q, x)$ of nome q and argument x .

See also

[jacobi_theta_4](#) for details.

Definition at line 5952 of file specfun.h.

8.3.2.217 jacobi_theta_4l()

```
long double __gnu_cxx::jacobi_theta_4l (
    long double __q,
    long double __x ) [inline]
```

Return the Jacobi theta-4 function $\theta_4(q, x)$ of nome q and argument x .

See also

[jacobi_theta_4](#) for details.

Definition at line 5962 of file specfun.h.

8.3.2.218 jacobi_zeta()

```
template<typename _Tk , typename _Tphi >
__gnu_cxx::fp_promote_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 4457 of file `specfun.h`.

8.3.2.219 `jacobi_zetaf()`

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

Definition at line 4432 of file `specfun.h`.

8.3.2.220 `jacobi_zetal()`

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

Definition at line 4436 of file `specfun.h`.

8.3.2.221 `jacobif()`

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

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

See also

[jacobi](#) for details.

Definition at line 2202 of file `specfun.h`.

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

8.3.2.222 `jacobil()`

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

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

See also

[jacobi](#) for details.

Definition at line 2216 of file specfun.h.

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

8.3.2.223 `lbinomial()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 4277 of file specfun.h.

8.3.2.224 lbinomialf()

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

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

See also

[lbinomial](#) for details.

Definition at line 4248 of file specfun.h.

8.3.2.225 lbinomiall()

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

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

See also

[lbinomial](#) for details.

Definition at line 4257 of file specfun.h.

8.3.2.226 ldouble_factorial()

```
template<typename _Tp>
__gnu_cxx::fp_promote_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 4191 of file specfun.h.

8.3.2.227 ldoube_factorialf()

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

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

See also

[ldouble_factorial](#) for more details

Definition at line 4164 of file specfun.h.

8.3.2.228 ldoube_factoriall()

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

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

See also

[double_factorial](#) for more details

Definition at line 4174 of file specfun.h.

8.3.2.229 legendre_q()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 digamma or psi 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 4367 of file `specfun.h`.

8.3.2.230 `legendre_qf()`

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

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

See also

[legendre_q](#) for details.

Definition at line 4333 of file `specfun.h`.

8.3.2.231 `legendre_ql()`

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

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

See also

[legendre_q](#) for details.

Definition at line 4343 of file `specfun.h`.

8.3.2.232 lerch_phi()

```
template<typename _Tp , typename _Ts , typename _Ta >
__gnu_cxx::fp_promote_t<_Tp, _Ts, _Ta> __gnu_cxx::lerch_phi (
    _Tp __z,
    _Ts __s,
    _Ta __a ) [inline]
```

Return the Lerch transcendent $\Phi(z, s, a)$.

The series is:

$$*\Phi(z, s, a) = \sum_{k=0}^{\infty} \frac{z^k}{(a + k^s)}$$

Definition at line 7018 of file specfun.h.

8.3.2.233 lerch_phif()

```
float __gnu_cxx::lerch_phif (
    float __z,
    float __s,
    float __a ) [inline]
```

Return the Lerch transcendent $\Phi(z, s, a)$ for float arguments.

See also

[lerch_phi](#) for details.

Definition at line 6995 of file specfun.h.

8.3.2.234 lerch_phil()

```
long double __gnu_cxx::lerch_phil (
    long double __z,
    long double __s,
    long double __a ) [inline]
```

Return the Lerch transcendent $\Phi(z, s, a)$ for long double arguments.

See also

[lerch_phi](#) for details.

Definition at line 7005 of file specfun.h.

8.3.2.235 `lfactorial()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 4149 of file `specfun.h`.

8.3.2.236 `lfactorialf()`

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

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

See also

[lfactorial](#) for more details

Definition at line 4127 of file `specfun.h`.

8.3.2.237 `lfactoriall()`

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

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

See also

[lfactorial](#) for more details

Definition at line 4137 of file `specfun.h`.

8.3.2.238 lfalling_factorial()

```
template<typename _Tp , typename _Tnu >
__gnu_cxx::fp_promote_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

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

In particular, $n^n = n!$. 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.

Definition at line 3953 of file specfun.h.

8.3.2.239 lfalling_factorialf()

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

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

See also

[lfalling_factorial](#) for details.

Definition at line 3918 of file specfun.h.

8.3.2.240 lfalling_factoriall()

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

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

See also

[lfalling_factorial](#) for details.

Definition at line 3928 of file specfun.h.

8.3.2.241 lgamma() [1/2]

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

Return the logarithm of the gamma function for real argument.

Definition at line 2937 of file specfun.h.

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

8.3.2.242 lgamma() [2/2]

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

Return the logarithm of the gamma function for complex argument.

Definition at line 2970 of file specfun.h.

8.3.2.243 lgammaf() [1/2]

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

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

See also

[lgamma](#) for details.

Definition at line 2919 of file specfun.h.

8.3.2.244 lgammaf() [2/2]

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

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

See also

[lgamma](#) for details.

Definition at line 2952 of file `specfun.h`.

8.3.2.245 lgammal() [1/2]

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

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

See also

[lgamma](#) for details.

Definition at line 2929 of file `specfun.h`.

8.3.2.246 lgammal() [2/2]

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

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

See also

[lgamma](#) for details.

Definition at line 2962 of file `specfun.h`.

8.3.2.247 logint()

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

Return the logarithmic integral of argument x .

The logarithmic integral is defined by

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

Parameters

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

Definition at line 1696 of file specfun.h.

8.3.2.248 logintf()

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

Return the logarithmic integral of argument x .

See also

[logint](#) for details.

Definition at line 1672 of file specfun.h.

8.3.2.249 logintl()

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

Return the logarithmic integral of argument x .

See also

[logint](#) for details.

Definition at line 1681 of file specfun.h.

8.3.2.250 logistic_p()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::fp_promote_t<_Ta, _Tb, _Tp> __gnu_cxx::logistic_p (
    _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 6767 of file specfun.h.

8.3.2.251 logistic_pdf()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::fp_promote_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 6750 of file specfun.h.

8.3.2.252 lognormal_p()

```
template<typename _Tmu , typename _Tsig , typename _Tp >
__gnu_cxx::fp_promote_t<_Tmu, _Tsig, _Tp> __gnu_cxx::lognormal_p (
    _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 6536 of file specfun.h.

8.3.2.253 `lognormal_pdf()`

```
template<typename _Tmu , typename _Tsig , typename _Tp >
__gnu_cxx::fp_promote_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 6519 of file `specfun.h`.

8.3.2.254 `lrising_factorial()`

```
template<typename _Tp , typename _Tnu >
__gnu_cxx::fp_promote_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 3903 of file `specfun.h`.

8.3.2.255 `lrising_factorialf()`

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

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

See also

[lrising_factorial](#) for details.

Definition at line 3869 of file `specfun.h`.

8.3.2.256 `lrising_factoriall()`

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

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

See also

[lrising_factorial](#) for details.

Definition at line 3879 of file `specfun.h`.

8.3.2.257 `normal_p()`

```
template<typename _Tmu , typename _Tsig , typename _Tp >
__gnu_cxx::fp_promote_t<_Tmu, _Tsig, _Tp> __gnu_cxx::normal_p (
    _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 6503 of file `specfun.h`.

8.3.2.258 `normal_pdf()`

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

Return the gamma cumulative propability distribution function.

The formula for the gamma probability density function is:

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

```
template<typename _Ta, typename _Tb, typename _Tp> inline __gnu_cxx::fp_promote_t<_Ta, _Tb, _Tp> gamma_
_p(_Ta __alpha, _Tb __beta, _Tp __x) { using __type = __gnu_cxx::fp_promote_t<_Ta, _Tb, _Tp>; return std::__
detail::__gamma_p<__type>(__alpha, __beta, __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 6486 of file `specfun.h`.

8.3.2.259 `owens_t()`

```
template<typename _Tph , typename _Tpa >
__gnu_cxx::fp_promote_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

<code>↔ _h</code>	The shape factor
<code>↔ _a</code>	The integration limit

Definition at line 6023 of file `specfun.h`.

8.3.2.260 `owens_tf()`

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

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

See also

[owens_t](#) for details.

Definition at line 5995 of file `specfun.h`.

8.3.2.261 `owens_tl()`

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

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

See also

[owens_t](#) for details.

Definition at line 6005 of file `specfun.h`.

8.3.2.262 polygamma()

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

Return the polygamma function of argument x .

The the polygamma or digamma function is defined by

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

Parameters

\leftrightarrow	The parameter
x	

Definition at line 3611 of file specfun.h.

8.3.2.263 polygammaf()

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

Return the polygamma function of `float` argument x .

See also

[polygamma](#) for details.

Definition at line 3585 of file specfun.h.

8.3.2.264 polygammal()

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

Return the polygamma function of `long double` argument x .

See also

[polygamma](#) for details.

Definition at line 3595 of file specfun.h.

8.3.2.265 polylog() [1/2]

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

Return the polylogarithm function of real order s and real argument w .

The polylogarithm function is defined by

$$Li_s(w) = \sum_{k=1}^{\infty} \frac{w^k}{k^s}$$

Parameters

\leftrightarrow _s	Order.
\leftrightarrow _w	Argument.

Definition at line 5049 of file specfun.h.

8.3.2.266 polylog() [2/2]

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

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

The polylogarithm function is defined by

$$Li_s(x) = \sum_{k=1}^{\infty} \frac{w^k}{k^s}$$

Parameters

\leftrightarrow _s	Order.
\leftrightarrow _w	Argument.

Definition at line 5089 of file specfun.h.

8.3.2.267 polylogf() [1/2]

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

Return the real polylogarithm function of real order s and real argument w .

See also

[polylog](#) for details.

Definition at line 5022 of file specfun.h.

8.3.2.268 polylogf() [2/2]

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

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

See also

[polylog](#) for details.

Definition at line 5062 of file specfun.h.

8.3.2.269 polylogl() [1/2]

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

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

See also

[polylog](#) for details.

Definition at line 5032 of file specfun.h.

8.3.2.270 `polylog()` [2/2]

```
std::complex<long double> __gnu_cxx::polylog1 (
    long double __s,
    std::complex< long double > __w ) [inline]
```

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

See also

[polylog](#) for details.

Definition at line 5072 of file specfun.h.

8.3.2.271 `radpoly()`

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

Parameters

<code>__rho</code>	The radial argument
--------------------	---------------------

Definition at line 2418 of file specfun.h.

8.3.2.272 `radpolyf()`

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

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

See also

[radpoly](#) for details.

Definition at line 2379 of file specfun.h.

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

8.3.2.273 `radpolyl()`

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

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

See also

[radpoly](#) for details.

Definition at line 2390 of file specfun.h.

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

8.3.2.274 rising_factorial()

```
template<typename _Tp , typename _Tnu >
__gnu_cxx::fp_promote_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 3996 of file specfun.h.

8.3.2.275 rising_factorialf()

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

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

See also

[rising_factorial](#) for details.

Definition at line 3968 of file specfun.h.

8.3.2.276 rising_factoriall()

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

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

See also

[rising_factorial](#) for details.

Definition at line 3978 of file specfun.h.

8.3.2.277 `sin_pi()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 6153 of file `specfun.h`.

8.3.2.278 `sin_pif()`

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

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

See also

[sin_pi](#) for more details.

Definition at line 6126 of file `specfun.h`.

8.3.2.279 `sin_pil()`

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

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

See also

[sin_pi](#) for more details.

Definition at line 6136 of file `specfun.h`.

8.3.2.280 sinc()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 1617 of file `specfun.h`.

8.3.2.281 sinc_pi()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 1658 of file `specfun.h`.

8.3.2.282 sinc_pif()

```
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 1632 of file specfun.h.

8.3.2.283 sinc_pil()

```
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 1642 of file specfun.h.

8.3.2.284 sincf()

```
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 1591 of file specfun.h.

8.3.2.285 `sinc()`

```
long double __gnu_cxx::sinc (
    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 1601 of file `specfun.h`.

8.3.2.286 `sincos()` [1/2]

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

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

See also

[sincos](#) for details.

Definition at line 6391 of file `specfun.h`.

8.3.2.287 `sincos()` [2/2]

```
template<typename _Tp >
__gnu_cxx::__sincos_t<__gnu_cxx::fp_promote_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 6402 of file `specfun.h`.

8.3.2.288 sincos_pi()

```
template<typename _Tp >
__gnu_cxx::__sincos_t<__gnu_cxx::fp_promote_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 6436 of file specfun.h.

8.3.2.289 sincos_pif()

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

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

See also

[sincos_pi](#) for details.

Definition at line 6414 of file specfun.h.

8.3.2.290 sincos_pil()

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

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

See also

[sincos_pi](#) for details.

Definition at line 6424 of file specfun.h.

8.3.2.291 sincosf()

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

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

Definition at line 6373 of file `specfun.h`.

8.3.2.292 sincosl()

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

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

See also

[sincos](#) for details.

Definition at line 6382 of file `specfun.h`.

8.3.2.293 sinh_pi()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 6195 of file specfun.h.

8.3.2.294 `sinh_pif()`

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

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

See also

[sinh_pi](#) for more details.

Definition at line 6168 of file specfun.h.

8.3.2.295 `sinh_pil()`

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

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

See also

[sinh_pi](#) for more details.

Definition at line 6178 of file specfun.h.

8.3.2.296 `sinhc()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 2500 of file `specfun.h`.

8.3.2.297 `sinhc_pi()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 2459 of file `specfun.h`.

8.3.2.298 `sinhc_pif()`

```
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 2433 of file specfun.h.

8.3.2.299 sinhc_pil()

```
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 2443 of file specfun.h.

8.3.2.300 sinhcf()

```
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 2474 of file specfun.h.

8.3.2.301 sinhcl()

```
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 2484 of file specfun.h.

8.3.2.302 `sinhint()`

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

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

The hyperbolic sine integral is defined by

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

Template Parameters

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

Parameters

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

Definition at line 1816 of file `specfun.h`.

8.3.2.303 `sinhintf()`

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

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

See also

[sinhint](#) for details.

Definition at line 1789 of file `specfun.h`.

8.3.2.304 `sinhintl()`

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

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

See also

[sinhint](#) for details.

Definition at line 1799 of file `specfun.h`.

8.3.2.305 sinint()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 __x	The real upper integration limit
--------------------------	----------------------------------

Definition at line 1735 of file specfun.h.

8.3.2.306 sinintf()

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

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

See also

[sinint](#) for details.

Definition at line 1710 of file specfun.h.

8.3.2.307 sinintl()

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

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

See also

[sinint](#) for details.

Definition at line 1720 of file specfun.h.

8.3.2.308 sph_bessel_i()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 2736 of file specfun.h.

8.3.2.309 sph_bessel_if()

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

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

See also

[sph_bessel_i](#) for details.

Definition at line 2707 of file specfun.h.

8.3.2.310 sph_bessel_il()

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

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

See also

[sph_bessel_i](#) for details.

Definition at line 2717 of file specfun.h.

8.3.2.311 sph_bessel_k()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 2780 of file specfun.h.

8.3.2.312 sph_bessel_kf()

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

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

See also

[sph_bessel_k](#) for more details.

Definition at line 2751 of file `specfun.h`.

8.3.2.313 sph_bessel_kl()

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

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

See also

[sph_bessel_k](#) for more details.

Definition at line 2761 of file `specfun.h`.

8.3.2.314 sph_hankel_1() [1/2]

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

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

Definition at line 2644 of file specfun.h.

8.3.2.315 `sph_hankel_1()` [2/2]

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

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

The spherical Hankel function of the first kind is defined by

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

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

Parameters

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

Definition at line 4906 of file specfun.h.

8.3.2.316 `sph_hankel_1f()` [1/2]

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

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

See also

[sph_hankel_1](#) for details.

Definition at line 2611 of file specfun.h.

8.3.2.317 sph_hankel_1f() [2/2]

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

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

See also

[sph_hankel_1](#) for more details.

Definition at line 4874 of file specfun.h.

8.3.2.318 sph_hankel_1l() [1/2]

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

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

See also

[sph_hankel_1](#) for details.

Definition at line 2621 of file specfun.h.

8.3.2.319 sph_hankel_1l() [2/2]

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

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

See also

[sph_hankel_1](#) for more details.

Definition at line 4885 of file specfun.h.

8.3.2.320 sph_hankel_2() [1/2]

```
template<typename _Tp >
std::complex<__gnu_cxx::fp_promote_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
<code>↵ _z</code>	The real argument

Definition at line 2692 of file specfun.h.

8.3.2.321 sph_hankel_2() [2/2]

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

$_n$	The integral order ≥ 0
$_x$	The complex argument

Definition at line 4954 of file specfun.h.

8.3.2.322 sph_hankel_2f() [1/2]

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

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

See also

[sph_hankel_2](#) for details.

Definition at line 2659 of file specfun.h.

8.3.2.323 sph_hankel_2f() [2/2]

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

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

See also

[sph_hankel_2](#) for more details.

Definition at line 4922 of file specfun.h.

8.3.2.324 sph_hankel_2l() [1/2]

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

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

See also

[sph_hankel_2](#) for details.

Definition at line 2669 of file specfun.h.

8.3.2.325 sph_hankel_2l() [2/2]

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

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

See also

[sph_hankel_2](#) for more details.

Definition at line 4933 of file specfun.h.

8.3.2.326 sph_harmonic()

```
template<typename _Ttheta , typename _Tphi >
std::complex<__gnu_cxx::fp_promote_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:

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

Note

$$Y_l^m(\theta, \phi) = 0 \text{ if } |m| > l.$$

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 5007 of file specfun.h.

8.3.2.327 sph_harmonicf()

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

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

See also

[sph_harmonic](#) for details.

Definition at line 4970 of file specfun.h.

8.3.2.328 sph_harmonic()

```
std::complex<long double> __gnu_cxx::sph_harmonic1 (
    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 4982 of file specfun.h.

8.3.2.329 stirling_1()

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

Return the Stirling number of the first kind.

The Stirling numbers of the first kind are the coefficients of the 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 6963 of file specfun.h.

8.3.2.330 `stirling_2()`

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

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

The series is:

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

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

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

Definition at line 6986 of file `specfun.h`.

8.3.2.331 `student_t_p()`

```
template<typename _Tt , typename _Tp >
__gnu_cxx::fp_promote_t<_Tp> __gnu_cxx::student_t_p (
    _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 6623 of file `specfun.h`.

8.3.2.332 student_t_pdf()

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

Return the complement of the Students T probability function.

The complement of the students T propability function is:

$$A_c(t|\nu) = I_{\frac{\nu}{\nu+t^2}}(\frac{\nu}{2}, \frac{1}{2}) = 1 - A(t|\nu)$$

Parameters

<code>__t</code>	
<code>__nu</code>	

Definition at line 6643 of file specfun.h.

8.3.2.333 tan_pi()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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 6321 of file specfun.h.

8.3.2.334 tan_pif()

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

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

See also

[tan_pi](#) for more details.

Definition at line 6294 of file specfun.h.

8.3.2.335 tan_pil()

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

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

See also

[tan_pi](#) for more details.

Definition at line 6304 of file specfun.h.

8.3.2.336 tanh_pi()

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

\longleftrightarrow	The argument
<code>__x</code>	

Definition at line 6363 of file specfun.h.

8.3.2.337 `tanh_pif()`

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

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

See also

[tanh_pi](#) for more details.

Definition at line 6336 of file specfun.h.

8.3.2.338 `tanh_pil()`

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

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

See also

[tanh_pi](#) for more details.

Definition at line 6346 of file specfun.h.

8.3.2.339 `tgamma()` [1/3]

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

Return the gamma function for real argument.

Definition at line 3002 of file specfun.h.

Referenced by `std::__detail::__tricomu_u_naive()`.

8.3.2.340 `tgamma()` [2/3]

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

Return the gamma function for complex argument.

Definition at line 3034 of file specfun.h.

8.3.2.341 `tgamma()` [3/3]

```
template<typename _Ta , typename _Tp >
__gnu_cxx::fp_promote_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 3071 of file specfun.h.

8.3.2.342 `tgamma_lower()`

```
template<typename _Ta , typename _Tp >
__gnu_cxx::fp_promote_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 3108 of file specfun.h.

8.3.2.343 `tgamma_lowerf()`

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

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

See also

[tgamma_lower](#) for details.

Definition at line 3086 of file `specfun.h`.

8.3.2.344 `tgamma_lowerl()`

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

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

See also

[tgamma_lower](#) for details.

Definition at line 3096 of file `specfun.h`.

8.3.2.345 `tgammaf()` [1/3]

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

Return the gamma function for `float` argument.

See also

[lgamma](#) for details.

Definition at line 2984 of file `specfun.h`.

8.3.2.346 `tgammaf()` [2/3]

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

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

See also

[lgamma](#) for details.

Definition at line 3016 of file `specfun.h`.

8.3.2.347 `tgammaf()` [3/3]

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

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

See also

[tgamma](#) for details.

Definition at line 3049 of file `specfun.h`.

8.3.2.348 `tgammal()` [1/3]

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

Return the gamma function for `long double` argument.

See also

[lgamma](#) for details.

Definition at line 2994 of file `specfun.h`.

8.3.2.349 `tgammal()` [2/3]

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

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

See also

[lgamma](#) for details.

Definition at line 3026 of file `specfun.h`.

8.3.2.350 `tgammal()` [3/3]

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

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

See also

[tgamma](#) for details.

Definition at line 3059 of file `specfun.h`.

8.3.2.351 `theta_1()`

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::fp_promote_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 exponential theta-1 function is defined by

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

Parameters

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

Definition at line 5449 of file specfun.h.

8.3.2.352 `theta_1f()`

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

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

See also

[theta_1](#) for details.

Definition at line 5421 of file specfun.h.

8.3.2.353 `theta_1l()`

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

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

See also

[theta_1](#) for details.

Definition at line 5431 of file specfun.h.

8.3.2.354 `theta_2()`

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::fp_promote_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 5492 of file specfun.h.

8.3.2.355 `theta_2f()`

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

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

See also

[theta_2](#) for details.

Definition at line 5464 of file specfun.h.

8.3.2.356 `theta_2l()`

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

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

See also

[theta_2](#) for details.

Definition at line 5474 of file specfun.h.

8.3.2.357 `theta_3()`

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::fp_promote_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 5535 of file specfun.h.

8.3.2.358 `theta_3f()`

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

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

See also

[theta_3](#) for details.

Definition at line 5507 of file specfun.h.

8.3.2.359 `theta_3l()`

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

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

See also

[theta_3](#) for details.

Definition at line 5517 of file specfun.h.

8.3.2.360 `theta_4()`

```
template<typename _Tpnu , typename _Tp >
__gnu_cxx::fp_promote_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 5578 of file specfun.h.

8.3.2.361 `theta_4f()`

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

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

See also

[theta_4](#) for details.

Definition at line 5550 of file specfun.h.

8.3.2.362 `theta_4l()`

```
long double __gnu_cxx::theta_4l (
    long double __nu,
    long double __x ) [inline]
```

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

See also

[theta_4](#) for details.

Definition at line 5560 of file specfun.h.

8.3.2.363 `theta_c()`

```
template<typename _Tp, typename _Tp>
__gnu_cxx::fp_promote_t<_Tp, _Tp> __gnu_cxx::theta_c (
    _Tp __k,
    _Tp __x ) [inline]
```

Return the Neville theta-c function $\theta_c(k, x)$ of modulus k and argument x .

The Neville theta-c function is defined by

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

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 5714 of file `specfun.h`.

8.3.2.364 `theta_cf()`

```
float __gnu_cxx::theta_cf (
    float __k,
    float __x ) [inline]
```

Return the Neville theta-c function $\theta_c(k, x)$ of modulus k and argument x .

See also

[theta_c](#) for details.

Definition at line 5682 of file `specfun.h`.

8.3.2.365 `theta_cl()`

```
long double __gnu_cxx::theta_cl (
    long double __k,
    long double __x ) [inline]
```

Return the Neville theta-c function $\theta_c(k, x)$ of long double modulus k and argument x .

See also

[theta_c](#) for details.

Definition at line 5692 of file specfun.h.

8.3.2.366 `theta_d()`

```
template<typename _Tp, typename _Tp2 >
__gnu_cxx::fp_promote_t<_Tp, _Tp2> __gnu_cxx::theta_d (
    _Tp __k,
    _Tp2 __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

\hookleftarrow <code>_k</code>	The modulus $-1 \leq k \leq +1$
\hookleftarrow <code>_x</code>	The argument

Definition at line 5761 of file specfun.h.

8.3.2.367 theta_df()

```
float __gnu_cxx::theta_df (
    float __k,
    float __x ) [inline]
```

Return the Neville theta-d function $\theta_d(k, x)$ of modulus k and argument x .

See also

[theta_d](#) for details.

Definition at line 5729 of file specfun.h.

8.3.2.368 theta_dl()

```
long double __gnu_cxx::theta_dl (
    long double __k,
    long double __x ) [inline]
```

Return the Neville theta-d function $\theta_d(k, x)$ of long double modulus k and argument x .

See also

[theta_d](#) for details.

Definition at line 5739 of file specfun.h.

8.3.2.369 theta_n()

```
template<typename _TpK , typename _Tp >
__gnu_cxx::fp_promote_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 5808 of file specfun.h.

8.3.2.370 `theta_nf()`

```
float __gnu_cxx::theta_nf (
    float __k,
    float __x ) [inline]
```

Return the Neville theta-n function $\theta_n(k, x)$ of modulus k and argument x .

See also

[theta_n](#) for details.

Definition at line 5776 of file specfun.h.

8.3.2.371 `theta_nl()`

```
long double __gnu_cxx::theta_nl (
    long double __k,
    long double __x ) [inline]
```

Return the Neville theta-n function $\theta_n(k, x)$ of long double modulus k and argument x .

See also

[theta_n](#) for details.

Definition at line 5786 of file specfun.h.

8.3.2.372 `theta_s()`

```
template<typename _Tp, typename _Tp>
__gnu_cxx::fp_promote_t<_Tp, _Tp> __gnu_cxx::theta_s (
    _Tp __k,
    _Tp __x ) [inline]
```

Return the Neville theta-s function $\theta_s(k, x)$ of modulus k and argument x .

The Neville theta-s function is defined by

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

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 5667 of file `specfun.h`.

8.3.2.373 `theta_sf()`

```
float __gnu_cxx::theta_sf (
    float __k,
    float __x ) [inline]
```

Return the Neville theta-s function $\theta_s(k, x)$ of modulus k and argument x .

See also

[theta_s](#) for details.

Definition at line 5635 of file `specfun.h`.

8.3.2.374 `theta_sl()`

```
long double __gnu_cxx::theta_sl (
    long double __k,
    long double __x ) [inline]
```

Return the Neville theta-s function $\theta_s(k, x)$ of long double modulus k and argument x .

See also

[theta_s](#) for details.

Definition at line 5645 of file specfun.h.

8.3.2.375 `tricoli_u()`

```
template<typename _Tpa , typename _Tpc , typename _Tp >
__gnu_cxx::fp_promote_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 numerator parameter a , denominator 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

\leftarrow <code>__a</code>	The numerator parameter
\leftarrow <code>__c</code>	The denominator parameter
\leftarrow <code>__x</code>	The argument

Definition at line 1481 of file specfun.h.

8.3.2.376 `tricomi_uf()`

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

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

See also

[tricomi_u](#) for details.

Definition at line 1447 of file `specfun.h`.

8.3.2.377 `tricomi_ul()`

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

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

See also

[tricomi_u](#) for details.

Definition at line 1458 of file `specfun.h`.

8.3.2.378 `weibull_p()`

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::fp_promote_t<_Ta, _Tb, _Tp> __gnu_cxx::weibull_p (
    _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 6603 of file `specfun.h`.

8.3.2.379 weibull_pdf()

```
template<typename _Ta , typename _Tb , typename _Tp >
__gnu_cxx::fp_promote_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 6587 of file specfun.h.

8.3.2.380 zernike()

```
template<typename _Trho , typename _Tphi >
__gnu_cxx::fp_promote_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 2363 of file `specfun.h`.

8.3.2.381 `zernikef()`

```
float __gnu_cxx::zernikef (
    unsigned int __n,
    int __m,
    float __rho,
    float __phi ) [inline]
```

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n , signed order m , and real radial argument ρ and azimuthal angle ϕ .

See also

[zernike](#) for details.

Definition at line 2324 of file `specfun.h`.

8.3.2.382 `zernikel()`

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

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n , signed order m , and real radial argument ρ and azimuthal angle ϕ .

See also

[zernike](#) for details.

Definition at line 2335 of file `specfun.h`.

Chapter 9

Namespace Documentation

9.1 `__gnu_cxx` Namespace Reference

Classes

- struct [__airy_t](#)
- struct [__chebyshev_t_t](#)
- struct [__chebyshev_u_t](#)
- struct [__chebyshev_v_t](#)
- struct [__chebyshev_w_t](#)
- struct [__cyl_bessel_t](#)
- struct [__cyl_coulomb_t](#)
- struct [__cyl_hankel_t](#)
- struct [__cyl_mod_bessel_t](#)
- struct [__fock_airy_t](#)
- struct [__fp_is_integer_t](#)
- struct [__gamma_inc_t](#)
- struct [__gamma_temme_t](#)

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

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

and

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

where $-1/2 \leq \mu \leq 1/2$ 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 [__gappa_pq_t](#)
- struct [__gegenbauer_t](#)
- struct [__hermite_he_t](#)
- struct [__hermite_t](#)
- struct [__jacobi_ellint_t](#)
- struct [__jacobi_t](#)
- struct [__laguerre_t](#)
- struct [__legendre_p_t](#)

- struct [__lgamma_t](#)
- struct [__quadrature_point_t](#)
- struct [__sincos_t](#)
- struct [__sph_bessel_t](#)
- struct [__sph_hankel_t](#)
- struct [__sph_mod_bessel_t](#)

Enumerations

- enum [gauss_quad_type](#) { [Gauss](#), [Gauss_Lobatto](#), [Gauss_Radau_lower](#), [Gauss_Radau_upper](#) }

Enumeration for differing types of Gauss quadrature. The `gauss_quad_type` is used to determine the boundary condition modifications applied to orthogonal polynomials for quadrature rules.

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::fp_promote_t](#)< [_Tp](#) > [airy_ai](#) ([_Tp](#) __x)
- template<typename [_Tp](#) >
std::complex< [__gnu_cxx::fp_promote_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::fp_promote_t](#)< [_Tp](#) > [airy_bi](#) ([_Tp](#) __x)
- template<typename [_Tp](#) >
std::complex< [__gnu_cxx::fp_promote_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::fp_promote_t](#)< [_Tp](#) > [bernoulli](#) (unsigned int __n)
- template<typename [_Tp](#) >
[_Tp](#) [bernoulli](#) (unsigned int __n, [_Tp](#) __x)
- float [bernoullif](#) (unsigned int __n)

- long double [bernoulli](#) (unsigned int __n)
- template<typename _Tp >
__gnu_cxx::fp_promote_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::fp_promote_t< _Tp > [binomial_p](#) (_Tp __p, unsigned int __n, unsigned int __k)
Return the binomial cumulative distribution function.
- template<typename _Tp >
__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > [clausen](#) (unsigned int __m, _Tp __x)
- template<typename _Tp >
std::complex< __gnu_cxx::fp_promote_t< _Tp > > [clausen](#) (unsigned int __m, std::complex< _Tp > __z)
- template<typename _Tp >
__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t< _Tk > [comp_ellint_d](#) (_Tk __k)
- float [comp_ellint_df](#) (float __k)
- long double [comp_ellint_dl](#) (long double __k)
- float [comp_ellint_rf](#) (float __x, float __y)
- long double [comp_ellint_rf](#) (long double __x, long double __y)
- template<typename _Tx, typename _Ty >
__gnu_cxx::fp_promote_t< _Tx, _Ty > [comp_ellint_rf](#) (_Tx __x, _Ty __y)
- float [comp_ellint_rg](#) (float __x, float __y)
- long double [comp_ellint_rg](#) (long double __x, long double __y)
- template<typename _Tx, typename _Ty >
__gnu_cxx::fp_promote_t< _Tx, _Ty > [comp_ellint_rg](#) (_Tx __x, _Ty __y)
- template<typename _Tpa, typename _Tpc, typename _Tp >
__gnu_cxx::fp_promote_t< _Tpa, _Tpc, _Tp > [conf_hyperg](#) (_Tpa __a, _Tpc __c, _Tp __x)
- template<typename _Tpc, typename _Tp >
__gnu_cxx::fp_promote_t< _Tpc, _Tp > [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::fp_promote_t< _Tp > [cos_pi](#) (_Tp __x)
- float [cos_pif](#) (float __x)
- long double [cos_pil](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::fp_promote_t< _Tp > [cosh_pi](#) (_Tp __x)
- float [cosh_pif](#) (float __x)
- long double [cosh_pil](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::fp_promote_t< _Tp > [coshint](#) (_Tp __x)
- float [coshintf](#) (float __x)
- long double [coshintl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::fp_promote_t< _Tp > [cosint](#) (_Tp __x)
- float [cosintf](#) (float __x)
- long double [cosintl](#) (long double __x)
- template<typename _Tpnu, typename _Tp >
std::complex< __gnu_cxx::fp_promote_t< _Tpnu, _Tp > > [cyl_hankel_1](#) (_Tpnu __nu, _Tp __z)
- template<typename _Tpnu, typename _Tp >
std::complex< __gnu_cxx::fp_promote_t< _Tpnu, _Tp > > [cyl_hankel_1](#) (std::complex< _Tpnu > __nu, std::complex< _Tp > __z)
- std::complex< float > [cyl_hankel_1f](#) (float __nu, float __z)
- std::complex< float > [cyl_hankel_1f](#) (std::complex< float > __nu, std::complex< float > __z)
- 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 > __z)
- template<typename _Tpnu, typename _Tp >
std::complex< __gnu_cxx::fp_promote_t< _Tpnu, _Tp > > [cyl_hankel_2](#) (_Tpnu __nu, _Tp __z)

- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_t< _Tp > dawson (_Tp __x)`
- `float dawsonf (float __x)`
- `long double dawsonl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_t< _Tp > digamma (_Tp __x)`
- `float digammaf (float __x)`
- `long double digammal (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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_betall (long double __s)`
- `template<typename _Tp >`
`_Tp dirichlet_eta (_Tp __s)`
- `float dirichlet_etaf (float __s)`
- `long double dirichlet_etall (long double __s)`
- `template<typename _Tp >`
`_Tp dirichlet_lambda (_Tp __s)`
- `float dirichlet_lambdaf (float __s)`
- `long double dirichlet_lambdall (long double __s)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tlam, _Tp > [exponential_p](#) (_Tlam __lambda, _Tp __x)

Return the exponential cumulative probability density function.

- `template<typename _Tlam, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tlam, _Tp > exponential_pdf (_Tlam __lambda, _Tp __x)`

Return the exponential probability density function.

- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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

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

In particular, $n^n = n!$.

- float [falling_factorialf](#) (float __a, float __nu)
- long double [falling_factoriall](#) (long double __a, long double __nu)
- `template<typename _Tps, typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_t< _Tp > fisher_f_p (_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::fp_promote_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::fp_promote_t< _Tp > fresnel_c (_Tp __x)`
- float [fresnel_cf](#) (float __x)
- long double [fresnel_cl](#) (long double __x)
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > fresnel_s (_Tp __x)`
- float [fresnel_sf](#) (float __x)
- long double [fresnel_sl](#) (long double __x)

- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Ta, _Tp > gamma_p (_Ta __a, _Tp __x)`

Return the gamma cumulative propability distribution function or the regularized lower incomplete gamma function.

- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Ta, _Tb, _Tp > gamma_pdf (_Ta __alpha, _Tb __beta, _Tp __x)`

Return the gamma propability distribution function.

- float [gamma_pf](#) (float __a, float __x)
- long double [gamma_pl](#) (long double __a, long double __x)

- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Ta, _Tp > gamma_q (_Ta __a, _Tp __x)`
Return the gamma complementary cumulative propability distribution (or survival) function or the regularized upper incomplete gamma function.
- `float gamma_qf (float __a, float __x)`
- `long double gamma_ql (long double __a, long double __x)`
- `template<typename _Ta >`
`__gnu_cxx::fp_promote_t< _Ta > gamma_reciprocal (_Ta __a)`
- `float gamma_reciprocalf (float __a)`
- `long double gamma_reciprocall (long double __a)`
- `template<typename _Tlam, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tlam, _Tp > gegenbauer (unsigned int __n, _Tlam __lambda, _Tp __x)`
- `float gegenbauerf (unsigned int __n, float __lambda, float __x)`
- `long double gegenbauerl (unsigned int __n, long double __lambda, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > harmonic (unsigned int __n)`
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Ta, _Tb, _Tp > ibeta (_Ta __a, _Tb __b, _Tp __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Ta, _Tb, _Tp > ibetac (_Ta __a, _Tb __b, _Tp __x)`
- `float ibetacf (float __a, float __b, float __x)`
- `long double ibetacL (long double __a, long double __b, long double __x)`
- `float ibetaf (float __a, float __b, float __x)`
- `long double ibetal (long double __a, long double __b, long double __x)`
- `template<typename _Talpha, typename _Tbeta, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Talpha, _Tbeta, _Tp > jacobi (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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 _Tpq, typename _Tp >
__gnu_cxx::fp_promote_t< _Tpq, _Tp > [jacobi_theta_1](#) (_Tpq __q, _Tp __x)
- float [jacobi_theta_1f](#) (float __q, float __x)
- long double [jacobi_theta_1l](#) (long double __q, long double __x)
- template<typename _Tpq, typename _Tp >
__gnu_cxx::fp_promote_t< _Tpq, _Tp > [jacobi_theta_2](#) (_Tpq __q, _Tp __x)
- float [jacobi_theta_2f](#) (float __q, float __x)
- long double [jacobi_theta_2l](#) (long double __q, long double __x)
- template<typename _Tpq, typename _Tp >
__gnu_cxx::fp_promote_t< _Tpq, _Tp > [jacobi_theta_3](#) (_Tpq __q, _Tp __x)
- float [jacobi_theta_3f](#) (float __q, float __x)
- long double [jacobi_theta_3l](#) (long double __q, long double __x)
- template<typename _Tpq, typename _Tp >
__gnu_cxx::fp_promote_t< _Tpq, _Tp > [jacobi_theta_4](#) (_Tpq __q, _Tp __x)
- float [jacobi_theta_4f](#) (float __q, float __x)
- long double [jacobi_theta_4l](#) (long double __q, long double __x)
- template<typename _Tk, typename _Tphi >
__gnu_cxx::fp_promote_t< _Tk, _Tphi > [jacobi_zeta](#) (_Tk __k, _Tphi __phi)
- float [jacobi_zetaf](#) (float __k, float __phi)
- long double [jacobi_zetal](#) (long double __k, long double __phi)
- float [jacobiif](#) (unsigned __n, float __alpha, float __beta, float __x)
- long double [jacobil](#) (unsigned __n, long double __alpha, long double __beta, long double __x)
- template<typename _Tp >
__gnu_cxx::fp_promote_t< _Tp > [lbinomial](#) (unsigned int __n, unsigned int __k)

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

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

The binomial coefficients are generated by:

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

- float [lbinomialf](#) (unsigned int __n, unsigned int __k)
- long double [lbinomiall](#) (unsigned int __n, unsigned int __k)
- template<typename _Tp >
__gnu_cxx::fp_promote_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::fp_promote_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, typename _Ts, typename _Ta >
__gnu_cxx::fp_promote_t< _Tp, _Ts, _Ta > [lerch_phi](#) (_Tp __z, _Ts __s, _Ta __a)

- float [lerch_phif](#) (float __z, float __s, float __a)
- long double [lerch_phil](#) (long double __z, long double __s, long double __a)
- template<typename _Tp >
__gnu_cxx::fp_promote_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::fp_promote_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

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

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.

- float [lfalling_factorialf](#) (float __a, float __nu)
- long double [lfalling_factoriall](#) (long double __a, long double __nu)
- template<typename _Ta >
__gnu_cxx::fp_promote_t< _Ta > [lgamma](#) (_Ta __a)
- template<typename _Ta >
std::complex< __gnu_cxx::fp_promote_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::fp_promote_t< _Tp > [logint](#) (_Tp __x)
- float [logintf](#) (float __x)
- long double [logintl](#) (long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::fp_promote_t< _Ta, _Tb, _Tp > [logistic_p](#) (_Ta __a, _Tb __b, _Tp __x)
- Return the logistic cumulative distribution function.*
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::fp_promote_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::fp_promote_t< _Tmu, _Tsig, _Tp > [lognormal_p](#) (_Tmu __mu, _Tsig __sigma, _Tp __x)
- Return the lognormal cumulative probability density function.*
- template<typename _Tmu, typename _Tsig, typename _Tp >
__gnu_cxx::fp_promote_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::fp_promote_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),

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

, 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::fp_promote_t< _Tmu, _Tsig, _Tp > [normal_p](#) (_Tmu __mu, _Tsig __sigma, _Tp __x)

Return the normal cumulative probability density function.

- template<typename _Tmu, typename _Tsig, typename _Tp >
__gnu_cxx::fp_promote_t< _Tmu, _Tsig, _Tp > [normal_pdf](#) (_Tmu __mu, _Tsig __sigma, _Tp __x)

Return the gamma cumulative probability distribution function.

- template<typename _Tph, typename _Tpa >
__gnu_cxx::fp_promote_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 _Tp >
__gnu_cxx::fp_promote_t< _Tp > [polygamma](#) (unsigned int __m, _Tp __x)
- float [polygammaf](#) (unsigned int __m, float __x)
- long double [polygammal](#) (unsigned int __m, long double __x)
- template<typename _Tp, typename _Wp >
__gnu_cxx::fp_promote_t< _Tp, _Wp > [polylog](#) (_Tp __s, _Wp __w)
- template<typename _Tp, typename _Wp >
std::complex< __gnu_cxx::fp_promote_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::fp_promote_t< _Tp > [radpoly](#) (unsigned int __n, unsigned int __m, _Tp __rho)
- float [radpolyf](#) (unsigned int __n, unsigned int __m, float __rho)
- long double [radpolyl](#) (unsigned int __n, unsigned int __m, long double __rho)
- template<typename _Tp, typename _Tnu >
__gnu_cxx::fp_promote_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

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

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_factoriall](#) (long double __a, long double __nu)

- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > sin_pi (_Tp __x)`
- `float sin_pif (float __x)`
- `long double sin_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > sinc (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_t< _Tp > > sincos (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< __gnu_cxx::fp_promote_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::fp_promote_t< _Tp > sinh_pi (_Tp __x)`
- `float sinh_pif (float __x)`
- `long double sinh_pil (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > sinhc (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > sinhc_pi (_Tp __x)`
- `float sinhc_pif (float __x)`
- `long double sinhc_pil (long double __x)`
- `float sinhcf (float __x)`
- `long double sinhcl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > sinhint (_Tp __x)`
- `float sinhintf (float __x)`
- `long double sinhintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > sinint (_Tp __x)`
- `float sinintf (float __x)`
- `long double sinintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > > sph_hankel_1 (unsigned int __n, _Tp __z)`

- `template<typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_t< _Tp > > sph_hankel_2` (unsigned int __n, _Tp __z)
- `template<typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > student_t_p` (_Tt __t, unsigned int __nu)
Return the Students T probability function.
- `template<typename _Tt, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > student_t_pdf` (_Tt __t, unsigned int __nu)
Return the complement of the Students T probability function.
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > tan_pi` (_Tp __x)
- `float tan_pif` (float __x)
- `long double tan_pil` (long double __x)
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > tanh_pi` (_Tp __x)
- `float tanh_pif` (float __x)
- `long double tanh_pil` (long double __x)
- `template<typename _Ta >`
`__gnu_cxx::fp_promote_t< _Ta > tgamma` (_Ta __a)
- `template<typename _Ta >`
`std::complex< __gnu_cxx::fp_promote_t< _Ta > > tgamma` (std::complex< _Ta > __a)
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Ta, _Tp > tgamma` (_Ta __a, _Tp __x)
- `template<typename _Ta, typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tpnu, _Tp > theta_4` (`_Tpnu __nu`, `_Tp __x`)
- `float theta_4f` (`float __nu`, `float __x`)
- `long double theta_4l` (`long double __nu`, `long double __x`)
- `template<typename _Tp, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp, _Tp > theta_c` (`_Tp __k`, `_Tp __x`)
- `float theta_cf` (`float __k`, `float __x`)
- `long double theta_cl` (`long double __k`, `long double __x`)
- `template<typename _Tp, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp, _Tp > theta_d` (`_Tp __k`, `_Tp __x`)
- `float theta_df` (`float __k`, `float __x`)
- `long double theta_dl` (`long double __k`, `long double __x`)
- `template<typename _Tp, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp, _Tp > theta_n` (`_Tp __k`, `_Tp __x`)
- `float theta_nf` (`float __k`, `float __x`)
- `long double theta_nl` (`long double __k`, `long double __x`)
- `template<typename _Tp, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp, _Tp > theta_s` (`_Tp __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::fp_promote_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::fp_promote_t< _Ta, _Tb, _Tp > weibull_p` (`_Ta __a`, `_Tb __b`, `_Tp __x`)
Return the Weibull cumulative probability density function.
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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 Enumeration Type Documentation

9.1.1.1 gauss_quad_type

```
enum __gnu_cxx::gauss_quad_type
```

Enumeration for differing types of Gauss quadrature. The `gauss_quad_type` is used to determine the boundary condition modifications applied to orthogonal polynomials for quadrature rules.

Enumerator

Gauss	Gauss quadrature.
Gauss_Lobatto	Gauss-Lobatto quadrature.
Gauss_Radau_lower	Gauss-Radau quadrature including the node -1.
Gauss_Radau_upper	Gauss-Radau quadrature including the node +1.

Definition at line 47 of file `specfun_state.h`.

9.1.2 Function Documentation

9.1.2.1 __fp_is_equal()

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

A function to reliably compare two floating point numbers.

Parameters

<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()`.

9.1.2.2 `__fp_is_even_integer()`

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

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

Parameters

<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()`.

9.1.2.3 `__fp_is_half_integer()`

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

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

Parameters

<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()`.

9.1.2.4 __fp_is_half_odd_integer()

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

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

Parameters

<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::__digamma()`.

9.1.2.5 __fp_is_integer()

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

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

Parameters

<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::__conf_hyperg()`, `std::__detail::__conf_hyperg_lim()`, `std::__detail::__digamma()`, `std::__detail::__dirichlet_eta()`, `std::__detail::__falling_factorial()`, `__fp_is_even_integer()`, `__fp_is_odd_integer()`, `std::__detail::__gamma()`, `std::__detail::__gamma_p()`, `std::__detail::__gamma_q()`, `std::__detail::__gamma_reciprocal()`, `std::__detail::__gamma_series()`, `std::__detail::__hyperg()`, `std::__detail::__hyperg_reflect()`, `std::__detail::__log_falling_factorial()`, `std::__detail::__log_gamma()`, `std::__detail::__polygamma()`, `std::__detail::__polylog()`, `std::__detail::__polylog_exp()`, `std::__detail::__riemann_zeta()`, `std::__detail::__riemann_zeta_m_1()`, `std::__detail::__tgamma()`, `std::__detail::__tgamma_lower()`, and `std::__detail::__tricoli_u_naive()`.

9.1.2.6 __fp_is_odd_integer()

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

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

Parameters

<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()`.

9.1.2.7 __fp_is_zero()

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

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

Parameters

<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 $\max(a, b) * \text{mul} * \text{epsilon}$

Definition at line 106 of file `math_util.h`.

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

9.1.2.8 __fp_max_abs()

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

A function to return the maximum 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()`.

9.1.2.9 __parity()

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

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

Definition at line 47 of file `math_util.h`.

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

9.2 std Namespace Reference

Namespaces

- [__detail](#)

Implementation-space details.

Functions

- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t](#)< _Tp > [expint](#) (_Tp __x)
- float [expintf](#) (float __x)
- long double [expintl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t](#)< _Tp > [riemann_zeta](#) (_Tp __s)
- float [riemann_zetaf](#) (float __s)
- long double [riemann_zetal](#) (long double __s)
- template<typename _Tp >
[__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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

Implementation-space details.

Classes

- struct [__gamma_lanczos_data](#)
- struct [__gamma_lanczos_data< double >](#)
- struct [__gamma_lanczos_data< float >](#)
- struct [__gamma_lanczos_data< long double >](#)
- struct [__gamma_spouge_data](#)
- struct [__gamma_spouge_data< double >](#)
- struct [__gamma_spouge_data< float >](#)
- struct [__gamma_spouge_data< long double >](#)
- struct [__jacobi_lattice_t](#)
- struct [__jacobi_theta_0_t](#)
- struct [__weierstrass_invariants_t](#)
- struct [__weierstrass_roots_t](#)
- class [_Airy](#)
- class [_Airy_asymp](#)
- struct [_Airy_asymp_data](#)
- struct [_Airy_asymp_data< double >](#)
- struct [_Airy_asymp_data< float >](#)
- struct [_Airy_asymp_data< long double >](#)
- class [_Airy_asymp_series](#)
- struct [_Airy_default_radai](#)
- struct [_Airy_default_radai< double >](#)
- struct [_Airy_default_radai< float >](#)
- struct [_Airy_default_radai< 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 degree n , order m : $L_n^{(m)}(x)$.
- `template<typename _Tp >`
`_Tp __assoc_legendre_p (unsigned int __l, unsigned int __m, _Tp __x, _Tp __phase=__Tp{+1})`
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_p (_Tp __a, _Tp __b, _Tp __x)`
- `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_p (_Tp __p, unsigned int __n, unsigned int __k)`
Return the 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 _Tp >`
`_Tp __binomial_q (_Tp __p, unsigned int __n, unsigned int __k)`
Return the complementary binomial cumulative distribution function.
- `template<typename _Sp, typename _Tp >`
`_Tp __bose_einstein (_Sp __s, _Tp __x)`
- `template<typename _Tp >`
`_Tp __cauchy_p (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`
`std::tuple< _Tp, _Tp, _Tp > __chebyshev_recur (unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1)`
- `template<typename _Tp >`
`__gnu_cxx::__chebyshev_t_t< _Tp > __chebyshev_t (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__chebyshev_u_t< _Tp > __chebyshev_u (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__chebyshev_v_t< _Tp > __chebyshev_v (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__chebyshev_w_t< _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 $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals as a pair.
- `template<typename _Tp >`
`void __chshint_cont_frac (_Tp __t, _Tp &_Chi, _Tp &_Shi)`
This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by continued fraction for positive argument.
- `template<typename _Tp >`
`void __chshint_series (_Tp __t, _Tp &_Chi, _Tp &_Shi)`

This function computes the hyperbolic cosine $\text{Chi}(x)$ and hyperbolic sine $\text{Shi}(x)$ integrals by series summation for positive argument.

- `template<typename _Tp >`
`std::complex< _Tp > __clamp_0_m2pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`std::complex< _Tp > __clamp_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`std::complex< _Tp > __clausen (unsigned int __m, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp __clausen (unsigned int __m, _Tp __x)`
- `template<typename _Tp >`
`_Tp __clausen_cl (unsigned int __m, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp __clausen_cl (unsigned int __m, _Tp __x)`
- `template<typename _Tp >`
`_Tp __clausen_sl (unsigned int __m, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp __clausen_sl (unsigned int __m, _Tp __x)`
- `template<typename _Tp >`
`_Tp __comp_ellint_1 (_Tp __k)`

Return the complete elliptic integral of the first kind $K(k)$ using the Carlson formulation.

- `template<typename _Tp >`
`_Tp __comp_ellint_2 (_Tp __k)`

Return the complete elliptic integral of the second kind $E(k)$ using the Carlson formulation.

- `template<typename _Tp >`
`_Tp __comp_ellint_3 (_Tp __k, _Tp __nu)`

Return the complete elliptic integral of the third kind $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$ using the Carlson formulation.

- `template<typename _Tp >`
`_Tp __comp_ellint_d (_Tp __k)`
- `template<typename _Tp >`
`_Tp __comp_ellint_rf (_Tp __x, _Tp __y)`
- `template<typename _Tp >`
`_Tp __comp_ellint_rg (_Tp __x, _Tp __y)`
- `template<typename _Tp >`
`_Tp __conf_hyperg (_Tp __a, _Tp __c, _Tp __x)`

Return the confluent hypergeometric function ${}_1F_1(a; c; x) = M(a, c, x)$.

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

Return the confluent hypergeometric limit function ${}_0F_1(-; c; x)$.

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

This routine returns the confluent hypergeometric limit function by series expansion.

- `template<typename _Tp >`
`_Tp __conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)`

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

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

This routine returns the confluent hypergeometric function by series expansion.

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

- `template<typename _Tp >`
`std::complex< _Tp > __cos_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp __cosh_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __cosh_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp __coshint (const _Tp __x)`
Return the hyperbolic cosine integral $Chi(x)$.
- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __coulomb_CF1 (unsigned int __l, _Tp __eta, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __coulomb_CF2 (unsigned int __l, _Tp __eta, _Tp __x)`
- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __coulomb_f_recur (unsigned int __l_min, unsigned int __k_max, _Tp __eta, _Tp __x, _Tp __F_l_max, _Tp __Fp_l_max)`
- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __coulomb_g_recur (unsigned int __l_min, unsigned int __k_max, _Tp __eta, _Tp __x, _Tp __G_l_min, _Tp __Gp_l_min)`
- `template<typename _Tp >`
`_Tp __coulomb_norm (unsigned int __l, _Tp __eta)`
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Bessel function.
- `template<typename _Tp >`
`_Tp __cyl_bessel_i (_Tp __nu, _Tp __x)`
Return the regular modified Bessel function of order ν : $I_\nu(x)$.
- `template<typename _Tp >`
`_Tp __cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)`
This routine returns the cylindrical Bessel functions of order ν : J_ν or I_ν by series expansion.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_ik (_Tp __nu, _Tp __x)`
Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_mod_bessel_t< _Tp, _Tp, _Tp > __cyl_bessel_ik_asymp (_Tp __nu, _Tp __x)`
This routine computes the asymptotic modified cylindrical Bessel and functions of order 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_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 __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_steel (_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 $H_n^{(2)}u(x)$.
- `template<typename _Tp >`
`std::complex<_Tp> __cyl_hankel_2 (std::complex<_Tp> __nu, std::complex<_Tp> __z)`
Return the complex cylindrical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex<_Tp> __cyl_neumann (std::complex<_Tp> __nu, std::complex<_Tp> __z)`
Return the complex cylindrical Neumann function.
- `template<typename _Tp >`
`_Tp __cyl_neumann_n (_Tp __nu, _Tp __x)`
Return the Neumann function of order ν : $N_\nu(x)$.
- `template<typename _Tp >`
`_Tp __dawson (_Tp __x)`
Return the Dawson integral, $F(x)$, for real argument x .
- `template<typename _Tp >`
`_Tp __dawson_cont_frac (_Tp __x)`
Compute the Dawson integral using a sampling theorem representation.
- `template<typename _Tp >`
`_Tp __dawson_series (_Tp __x)`
Compute the Dawson integral using the series expansion.
- `template<typename _Tp >`
`_Tp __debye (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`void __debye_region (std::complex<_Tp> __alpha, int &__indexr, char &__aorb)`
- `template<typename _Tp >`
`_Tp __digamma (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 __digamma (_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 __digamma_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 __digamma_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 __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 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __eulerian_1_recur (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __eulerian_2 (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __eulerian_2_recur (unsigned int __n, unsigned int __m)`
- `template<typename _Tp >`
`_Tp __exp2 (_Tp __x)`
- `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_p (_Tp __lambda, _Tp __x)`
Return the exponential cumulative probability density function.
- `template<typename _Tp >`
`_Tp __exponential_pdf (_Tp __lambda, _Tp __x)`
Return the exponential probability density function.
- `template<typename _Tp >`
`_Tp __exponential_q (_Tp __lambda, _Tp __x)`
Return the complement of the exponential cumulative 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_p (_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 >
_Tp __fisher_f_q (_Tp __F, unsigned int __nu1, unsigned int __nu2)

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

- template<typename _Tp >
__gnu_cxx::__fock_airy_t<_Tp, std::complex<_Tp>> __fock_airy (_Tp __x)

Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w'_1(x)$ and $w'_2(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$

- template<typename _Tp >
std::complex<_Tp> __fresnel (const _Tp __x)

Return the Fresnel cosine and sine integrals as a complex number $\$f[C(x) + iS(x) \$f]$.

- template<typename _Tp >
void __fresnel_cont_frac (const _Tp __ax, _Tp &Cf, _Tp &Sf)

This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.

- template<typename _Tp >
void __fresnel_series (const _Tp __ax, _Tp &Cf, _Tp &Sf)

This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.

- template<typename _Tp >
_Tp __gamma (_Tp __a)

Return the gamma function $\Gamma(a)$. The gamma function is defined by:

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

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __gamma (_Tp __a, _Tp __x)`

Return the incomplete gamma functions.

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __gamma_cont_frac (_Tp __a, _Tp __x)`

Return the incomplete gamma function by continued fraction.

- `template<typename _Tp >`
`_Tp __gamma_p (_Tp __alpha, _Tp __beta, _Tp __x)`

Return the gamma cumulative propability distribution function.

- `template<typename _Tp >`
`_Tp __gamma_p (_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 __gamma_pdf (_Tp __alpha, _Tp __beta, _Tp __x)`

Return the gamma propability distribution function.

- `template<typename _Tp >`
`_Tp __gamma_q (_Tp __alpha, _Tp __beta, _Tp __x)`

Return the gamma complementary cumulative propability distribution function.

- `template<typename _Tp >`
`_Tp __gamma_q (_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 __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 >`
`__gnu_cxx::__gegenbauer_t< _Tp > __gegenbauer_recur (unsigned int __n, _Tp __lambda, _Tp __x)`
- `template<typename _Tp >`
`std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > __gegenbauer_zeros (unsigned int __n, _Tp __lambda)`
- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel←_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn)`
- `template<typename _Tp >`
`void __hankel_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__nup2, std::complex< _Tp > &__num2, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__num4d3, std::complex< _Tp > &__zeta, std::complex< _Tp > &__zetaphf, std::complex< _Tp > &__zetamhf, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__zetrat)`

Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.

- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel←_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z)`

This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.

- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > __hankel←_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z)`

Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order nu along with their derivatives.

- `template<typename _Tp >`
`void __hankel_uniform_outer (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__etm3h, std::complex< _Tp > &__etrat, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp > &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)`

Compute outer factors and associated functions of z and nu appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of z and nu returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.

- template<typename _Tp >
 void [__hankel_uniform_sum](#) (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > __num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > __Aip, std::complex< _Tp > __o4dp, std::complex< _Tp > __Aim, std::complex< _Tp > __o4dm, std::complex< _Tp > __od2p, std::complex< _Tp > __od0dp, std::complex< _Tp > __od2m, std::complex< _Tp > __od0dm, _Tp __eps, std::complex< _Tp > &__H1sum, std::complex< _Tp > &__H1psum, std::complex< _Tp > &__H2sum, std::complex< _Tp > &__H2psum)
Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to nterms (less than 5) to achieve relative error eps.
- template<typename _Tp >
 _Tp [__harmonic_number](#) (unsigned int __n)
- template<typename _Tp >
 _Tp [__hermite](#) (unsigned int __n, _Tp __x)
This routine returns the Hermite polynomial of order n: $H_n(x)$.
- template<typename _Tp >
 _Tp [__hermite_asymp](#) (unsigned int __n, _Tp __x)
This routine returns the Hermite polynomial of large order n: $H_n(x)$. We assume here that $x \geq 0$.
- template<typename _Tp >
[__gnu_cxx::__hermite_t](#)< _Tp > [__hermite_recur](#) (unsigned int __n, _Tp __x)
This routine returns the Hermite polynomial of order n: $H_n(x)$ by recursion on n.
- template<typename _Tp >
 std::vector< [__gnu_cxx::__quadrature_point_t](#)< _Tp > > [__hermite_zeros](#) (unsigned int __n, _Tp __proto=[__Tp{}](#))
- template<typename _Tp >
 _Tp [__heuman_lambda](#) (_Tp __k, _Tp __phi)
- template<typename _Tp >
 _Tp [__hurwitz_zeta](#) (_Tp __s, _Tp __a)
Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ 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_recur](#) (int __m, _Tp __b, _Tp __c, _Tp __x)
Return the hypergeometric polynomial ${}_2F_1(-m, b; c; x)$ by Holm recursion.
- 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_ellint_t< _Tp > __jacobi_ellint (_Tp __k, _Tp __u)`
- `template<typename _Tp >`
`__gnu_cxx::__jacobi_t< _Tp > __jacobi_recur (unsigned int __n, _Tp __alpha1, _Tp __beta1, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __jacobi_theta_1 (std::complex< _Tp > __q, std::complex< _Tp > __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_1 (_Tp __q, const _Tp __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_1_prod (_Tp __q, _Tp __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_1_sum (_Tp __q, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __jacobi_theta_2 (std::complex< _Tp > __q, std::complex< _Tp > __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_2 (_Tp __q, const _Tp __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_2_prod (_Tp __q, _Tp __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_2_sum (_Tp __q, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __jacobi_theta_3 (std::complex< _Tp > __q, std::complex< _Tp > __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_3 (_Tp __q, const _Tp __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_3_prod (_Tp __q, _Tp __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_3_sum (_Tp __q, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > __jacobi_theta_4 (std::complex< _Tp > __q, std::complex< _Tp > __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_4 (_Tp __q, const _Tp __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_4_prod (_Tp __q, _Tp __x)`
- `template<typename _Tp >`
`_Tp __jacobi_theta_4_sum (_Tp __q, _Tp __x)`
- `template<typename _Tp >`
`std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > __jacobi_zeros (unsigned int __n, _Tp __alpha1, _Tp __beta1)`
- `template<typename _Tp >`
`_Tp __jacobi_zeta (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`
`_Tp __kolmogorov_p (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tpa, typename _Tp >`
`_Tp __laguerre (unsigned int __n, _Tpa __alpha1, _Tp __x)`
This routine returns the associated Laguerre polynomial of degree n, order α : $L_n^{(\alpha)}(x)$.
- `template<typename _Tp >`
`_Tp __laguerre (unsigned int __n, _Tp __x)`
This routine returns the Laguerre polynomial of degree n: $L_n(x)$.

- `template<typename _Tpa, typename _Tp >`
`_Tp __laguerre_hyperg` (unsigned int __n, _Tpa __alpha1, _Tp __x)
Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.
- `template<typename _Tpa, typename _Tp >`
`_Tp __laguerre_large_n` (unsigned __n, _Tpa __alpha1, _Tp __x)
This routine returns the associated Laguerre polynomial of degree n, order $\alpha > -1$ for large n. Abramowitz & Stegun, 13.5.21.
- `template<typename _Tpa, typename _Tp >`
`__gnu_cxx::__laguerre_t<_Tpa, _Tp> __laguerre_recur` (unsigned int __n, _Tpa __alpha1, _Tp __x)
This routine returns the associated Laguerre polynomial of degree n, order $\alpha: L_n^{(\alpha)}(x)$ by recursion.
- `template<typename _Tp >`
`std::vector< __gnu_cxx::__quadrature_point_t<_Tp> > __laguerre_zeros` (unsigned int __n, _Tp __alpha1)
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __lanczos_binet1p` (_Tp __z)
Return the Binet function $J(1+z)$ by the Lanczos method. The Binet function is the log of the scaled Gamma function $\log(\Gamma^(z))$ defined by*

$$J(z) = \log(\Gamma^*(z)) = \log(\Gamma(z)) + z - \left(z - \frac{1}{2}\right) \log(z) - \log(2\pi)$$

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 >`
`__gnu_cxx::__legendre_p_t<_Tp> __legendre_p` (unsigned int __l, _Tp __x)
Return the Legendre polynomial by upward recursion on degree l.
- `template<typename _Tp >`
`_Tp __legendre_q` (unsigned int __l, _Tp __x)
Return the Legendre function of the second kind by upward recursion on degree 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 __nu)`

- `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_p (_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_p (_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_p (_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 >`
`std::complex< _Tp > __polar_pi (_Tp __rho, _Tp __phi_pi)`
- `template<typename _Tp >`
`std::complex< _Tp > __polar_pi (_Tp __rho, const std::complex< _Tp > &__phi_pi)`
- `template<typename _Tp >`
`_Tp __polygamma (unsigned int __m, _Tp __x)`
Return the polygamma function $\psi^{(m)}(x)$.

- `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::fp_promote_t< std::complex< _Tp >, _ArgType > __polylog_exp (_Tp __s, _ArgType __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg (int __n, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg_int (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg_int (int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg_real (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg_real (_Tp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos_int (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos_int (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos_real (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos_real (_Tp __s, _Tp __w)`
- `template<typename _PowTp, typename _Tp >`
`_Tp __polylog_exp_sum (_PowTp __s, _Tp __w)`
- `template<typename _Tp >`
`__gnu_cxx::__hermite_he_t< _Tp > __prob_hermite_recur (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 __radial_jacobi (unsigned int __n, unsigned int __m, _Tp __rho)`
- `template<typename _Tp >`
`std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > __radial_jacobi_zeros (unsigned int __n, unsigned int __m)`
- `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_laurent (_Tp __s)`

Compute the Riemann zeta function $\zeta(s)$ by Laurent expansion about $s = 1$.

- `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),

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

, 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),

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

, 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::fp_promote_t< _Tp > __sinc (_Tp __x)`

Return the sinus cardinal function

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

.

- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_t< _Tp > __sinhc (_Tp __x)`

Return the hyperbolic sinus cardinal function

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

- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > __sinhc_pi (_Tp __x)`

Return the reperiodized hyperbolic sinus cardinal function

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

- `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)`
Helper to compute complex spherical Bessel 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_p(_Tp __t, unsigned int __nu)`
Return 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 __student_t_q(_Tp __t, unsigned int __nu)`
Return the complement of the Students T probability function.
- `template<typename _Tp >`
`_Tp __tan_pi(_Tp __x)`
- `template<typename _Tp >`
`std::complex<_Tp> __tan_pi(std::complex<_Tp> __z)`
- `template<typename _Tp >`
`_Tp __tanh_pi(_Tp __x)`
- `template<typename _Tp >`
`std::complex<_Tp> __tanh_pi(std::complex<_Tp> __z)`
- `template<typename _Tp >`
`_Tp __tgamma(_Tp __a, _Tp __x)`

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

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

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

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

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

- `template<typename _Tp >`
`_Tp __theta_1 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_2 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_2_asymp (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_2_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_3 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_3_asymp (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_3_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_4 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_c (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_d (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_n (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_s (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __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_p (_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::fp_promote_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 size_t _Num_Stieltjes = 21`
- `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_number]`
- `constexpr unsigned long long _S_harmonic_number [_S_num_harmonic_number]`
- `constexpr _Factorial_table< long double > _S_neg_double_factorial_table [999]`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_double_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_double_factorials< double > = 301`
- `template<>`
`constexpr std::size_t _S_num_double_factorials< float > = 57`
- `template<>`
`constexpr std::size_t _S_num_double_factorials< long double > = 301`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_factorials< double > = 171`
- `template<>`
`constexpr std::size_t _S_num_factorials< float > = 35`
- `template<>`
`constexpr std::size_t _S_num_factorials< long double > = 171`
- `constexpr unsigned long long _S_num_harmonic_number = 29`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_neg_double_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< double > = 150`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< float > = 27`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< long double > = 999`
- `constexpr size_t _S_num_zetam1 = 121`
- `constexpr long double _S_Stieltjes [_Num_Stieltjes]`
- `constexpr long double _S_zetam1 [_S_num_zetam1]`

9.3.1 Detailed Description

Implementation-space details.

9.3.2 Function Documentation

9.3.2.1 `__airy()`

```
template<typename _Tp >
__gnu_cxx::__airy_t<_Tp, _Tp> std::__detail::__airy (
    _Tp __z )
```

Compute the Airy functions $Ai(x)$ and $Bi(x)$ and their first derivatives $Ai'(x)$ and $Bi'(x)$ respectively.

Parameters

<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 475 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`, and `__cyl_bessel_jn()`.

Referenced by `__airy_ai()`, `__airy_bi()`, `__fock_airy()`, and `__hermite_asymp()`.

9.3.2.2 `__airy_ai()`

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__airy_ai (
    std::complex<_Tp > __z )
```

Return the complex Airy Ai function.

Definition at line 2628 of file `sf_airy.tcc`.

References `__airy()`.

9.3.2.3 `__airy_arg()`

```
template<typename _Tp >
void std::__detail::__airy_arg (
    std::complex<_Tp > __num2d3,
    std::complex<_Tp > __zeta,
    std::complex<_Tp > & __argp,
    std::complex<_Tp > & __argm )
```

Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.

Parameters

in	<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 214 of file `sf_hankel.tcc`.

Referenced by `__hankel_uniform_outer()`.

9.3.2.4 `__airy_bi()`

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__airy_bi (
    std::complex< _Tp > __z )
```

Return the complex Airy Bi function.

Definition at line 2640 of file `sf_airy.tcc`.

References `__airy()`.

9.3.2.5 `__assoc_laguerre()`

```
template<typename _Tp >
_Tp std::__detail::__assoc_laguerre (
    unsigned int __n,
    unsigned int __m,
    _Tp __x )
```

This routine returns the associated Laguerre polynomial of degree n, order m: $L_n^{(m)}(x)$.

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

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

where the Laguerre polynomial is defined by:

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

Template Parameters

<code>_Tp</code>	The type of the parameter
------------------	---------------------------

Parameters

<code>_↔ _n</code>	The degree
<code>_↔ _m</code>	The order
<code>_↔ _x</code>	The argument

Returns

The value of the associated Laguerre polynomial of order n , degree m , and argument x .

Definition at line 366 of file `sf_laguerre.tcc`.

Referenced by `__hydrogen()`.

9.3.2.6 `__assoc_legendre_p()`

```
template<typename _Tp >
_Tp std::__detail::__assoc_legendre_p (
    unsigned int __l,
    unsigned int __m,
    _Tp __x,
    _Tp __phase = _Tp{+1} )
```

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:

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

Note

The Condon-Shortley phase factor $(-1)^m$ is absent by default.

$P_l^m(x) = 0$ if $m > l$.

Parameters

<code>__l</code>	The degree of the associated Legendre function. $l \geq 0$.	Generated by Doxygen
<code>__m</code>	The order of the associated Legendre function.	
<code>__x</code>	The argument of the associated Legendre function.	
<code>__phase</code>	The phase of the associated Legendre function. Use -1 for the Condon-Shortley phase convention.	

Definition at line 199 of file sf_legendre.tcc.

References `__legendre_p()`.

9.3.2.7 `__bernoulli()` [1/2]

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (
    unsigned int __n )
```

This returns Bernoulli number B_n .

Parameters

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

Returns

The Bernoulli number of order n.

Definition at line 128 of file sf_bernoulli.tcc.

Referenced by `__euler()`, and `__gnu_cxx::bernoulli()`.

9.3.2.8 `__bernoulli()` [2/2]

```
template<typename _Tp >
_Tp std::__detail::__bernoulli (
    unsigned int __n,
    _Tp __x )
```

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

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

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

The derivative is proportional to the previous polynomial:

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

The series expansion is:

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

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()`.

9.3.2.9 __bernoulli_2n()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (
    unsigned int __n )
```

This returns Bernoulli number B_{2n} at even integer arguments $2n$.

Parameters

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

Returns

The Bernoulli number of order $2n$.

Definition at line 140 of file sf_bernoulli.tcc.

9.3.2.10 __bernoulli_series()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series (
    unsigned int __n )
```

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

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

.

Note that

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

are small and rapidly decreasing 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.

9.3.2.11 __beta()

```
template<typename _Tp >
_Tp std::__detail::__beta (
    _Tp __a,
    _Tp __b )
```

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

The beta function is defined by

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

Parameters

\longleftrightarrow __a	The first argument of the beta function.
\longleftrightarrow __b	The second argument of the beta function.

Returns

The beta function.

Definition at line 215 of file sf_beta.tcc.

References __beta_gamma(), and __beta_lgamma().

Referenced by __fisher_f_pdf(), __gnu_cxx::gamma_pdf(), __gnu_cxx::jacobi(), __gnu_cxx::jacobif(), __gnu_cxx::jacobil(), and std::__detail::_Airy<_Tp>::operator().

9.3.2.12 __beta_gamma()

```
template<typename _Tp >
_Tp std::__detail::__beta_gamma (
    _Tp __a,
    _Tp __b )
```

Return the beta function: $B(a, b)$.

The beta function is defined by

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

Parameters

\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 77 of file sf_beta.tcc.

References `__gamma()`.

Referenced by `__beta()`.

9.3.2.13 `__beta_inc()`

```
template<typename _Tp >
_Tp std::__detail::__beta_inc (
    _Tp __a,
    _Tp __b,
    _Tp __x )
```

Return the regularized incomplete beta function, $I_x(a, b)$, of arguments a , b , and x .

The regularized incomplete beta function is defined by:

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

where

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

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

Parameters

\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 `__beta_p()`, `__binomial_p()`, `__binomial_q()`, `__fisher_f_p()`, `__fisher_f_q()`, `__student_t_p()`, and `__student_t_q()`.

9.3.2.14 `__beta_lgamma()`

```
template<typename _Tp >
_Tp std::__detail::__beta_lgamma (
    _Tp __a,
    _Tp __b )
```

Return the beta function $B(a, b)$ using the log gamma functions.

The beta function is defined by

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

Parameters

<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 125 of file `sf_beta.tcc`.

References `__log_gamma()`, and `__log_gamma_sign()`.

Referenced by `__beta()`.

9.3.2.15 `__beta_p()`

```
template<typename _Tp >
_Tp std::__detail::__beta_p (
    _Tp __a,
    _Tp __b,
    _Tp __x )
```

Definition at line 705 of file `sf_distributions.tcc`.

References `__beta_inc()`.

9.3.2.16 __beta_product()

```
template<typename _Tp >
_Tp std::__detail::__beta_product (
    _Tp __a,
    _Tp __b )
```

Return the beta function $B(x, y)$ using the product form.

The beta function is defined by

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

Here, we employ the product form:

$$B(a, b) = \frac{a+b}{ab} \prod_{k=1}^{\infty} \frac{1 + (a+b)/k}{(1+a/k)(1+b/k)} = \frac{a+b}{ab} \prod_{k=1}^{\infty} \left[1 - \frac{ab}{(a+k)(b+k)} \right]$$

Parameters

\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 179 of file sf_beta.tcc.

9.3.2.17 __binomial() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__binomial (
    unsigned int __n,
    unsigned int __k )
```

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

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

The binomial coefficients are generated by:

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

.

Parameters

\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 2538 of file sf_gamma.tcc.

References std::__detail::_Factorial_table<_Tp>::__n.

Referenced by __bernoulli().

9.3.2.18 __binomial() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__binomial (
    _Tp __nu,
    unsigned int __k )
```

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

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

The binomial coefficients are generated by:

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

.

Parameters

__nu	The real first argument of the binomial coefficient.
__k	The second argument of the binomial coefficient.

Returns

The binomial coefficient.

Definition at line 2598 of file sf_gamma.tcc.

References __gamma(), __log_binomial(), __log_binomial_sign(), and std::__detail::_Factorial_table<_Tp>::__n.

9.3.2.19 `__binomial_p()`

```
template<typename _Tp >
_Tp std::__detail::__binomial_p (
    _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

\longleftrightarrow <code>__p</code>	
\longleftrightarrow <code>__n</code>	
\longleftrightarrow <code>__k</code>	

Definition at line 614 of file `sf_distributions.tcc`.

References `__beta_inc()`.

9.3.2.20 `__binomial_pdf()`

```
template<typename _Tp >
_Tp std::__detail::__binomial_pdf (
    _Tp __p,
    unsigned int __n,
    unsigned int __k )
```

Return the binomial probability mass function.

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

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

Parameters

\longleftrightarrow <code>__p</code>	
\longleftrightarrow <code>__n</code>	
\longleftrightarrow <code>__k</code>	

Definition at line 578 of file sf_distributions.tcc.

9.3.2.21 __binomial_q()

```
template<typename _Tp >
_Tp std::__detail::__binomial_q (
    _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

\overleftarrow{p}	
\overleftarrow{n}	
\overleftarrow{k}	

Definition at line 644 of file sf_distributions.tcc.

References `__beta_inc()`.

9.3.2.22 __bose_einstein()

```
template<typename _Sp , typename _Tp >
_Tp std::__detail::__bose_einstein (
    _Sp __s,
    _Tp __x )
```

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

See also

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

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

Parameters

\leftrightarrow _s	The order $s \geq 0$.
\leftrightarrow _x	The real argument.

Returns

The real Bose-Einstein integral $G_s(x)$,

Definition at line 1466 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.23 `__cauchy_p()`

```
template<typename _Tp >
_Tp std::__detail::__cauchy_p (
    _Tp __a,
    _Tp __b,
    _Tp __x )
```

Definition at line 697 of file sf_distributions.tcc.

9.3.2.24 `__chebyshev_recur()`

```
template<typename _Tp >
std::tuple<_Tp, _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 $-1 \leq x \leq +1$
_C0	The value of the zeroth-order Chebyshev polynomial at x
_C1	The value of the first-order Chebyshev polynomial at x

Definition at line 60 of file sf_chebyshev.tcc.

Referenced by __chebyshev_t(), __chebyshev_u(), __chebyshev_v(), and __chebyshev_w().

9.3.2.25 __chebyshev_t()

```
template<typename _Tp >
__gnu_cxx::__chebyshev_t_t<_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

_Tp	The real type of the argument
-----	-------------------------------

Parameters

\leftrightarrow _n	The non-negative integral order
\leftrightarrow _x	The real argument $-1 \leq x \leq +1$

Definition at line 88 of file sf_chebyshev.tcc.

References __chebyshev_recur().

9.3.2.26 `__chebyshev_u()`

```
template<typename _Tp >
__gnu_cxx::__chebyshev_u_t<_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 118 of file `sf_chebyshev.tcc`.

References `__chebyshev_recur()`.

9.3.2.27 `__chebyshev_v()`

```
template<typename _Tp >
__gnu_cxx::__chebyshev_v_t<_Tp> std::__detail::__chebyshev_v (
    unsigned int __n,
    _Tp __x )
```

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

The Chebyshev polynomial of the third kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 149 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

9.3.2.28 `__chebyshev_w()`

```
template<typename _Tp >
__gnu_cxx::__chebyshev_w_t<_Tp> std::__detail::__chebyshev_w (
    unsigned int __n,
    _Tp __x )
```

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

The Chebyshev polynomial of the fourth kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 180 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

9.3.2.29 `__chi_squared_pdf()`

```
template<typename _Tp >
_Tp std::__detail::__chi_squared_pdf (
    _Tp __chi2,
    unsigned int __nu )
```

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

The chi-squared propability function is related to the normalized lower incomplete gamma function:

$$P(\chi^2|\nu) = \Gamma_P\left(\frac{\nu}{2}, \frac{\chi^2}{2}\right)$$

Definition at line 75 of file `sf_distributions.tcc`.

References `__gamma_p()`.

9.3.2.30 `__chi_squared_pdfc()`

```
template<typename _Tp >
_Tp std::__detail::__chi_squared_pdfc (
    _Tp __chi2,
    unsigned int __nu )
```

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

The complementary chi-squared propability function is related to the normalized upper incomplete gamma function:

$$Q(\chi^2|\nu) = \Gamma_Q\left(\frac{\nu}{2}, \frac{\chi^2}{2}\right)$$

Definition at line 99 of file `sf_distributions.tcc`.

References `__gamma_q()`.

9.3.2.31 __chshint()

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__chshint (
    _Tp __x,
    _Tp & __Chi,
    _Tp & __Shi )
```

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

The hyperbolic cosine integral is defined by:

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

The hyperbolic sine integral is defined by:

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

Definition at line 166 of file sf_hypint.tcc.

References `__chshint_cont_frac()`, and `__chshint_series()`.

9.3.2.32 __chshint_cont_frac()

```
template<typename _Tp >
void std::__detail::__chshint_cont_frac (
    _Tp __t,
    _Tp & __Chi,
    _Tp & __Shi )
```

This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by continued fraction for positive argument.

Definition at line 53 of file sf_hypint.tcc.

Referenced by `__chshint()`.

9.3.2.33 __chshint_series()

```
template<typename _Tp >
void std::__detail::__chshint_series (
    _Tp __t,
    _Tp & __Chi,
    _Tp & __Shi )
```

This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by series summation for positive argument.

Definition at line 96 of file sf_hypint.tcc.

Referenced by `__chshint()`.

9.3.2.34 `__clamp_0_m2pi()`

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__clamp_0_m2pi (
    std::complex< _Tp > __z )
```

Definition at line 184 of file `sf_polylog.tcc`.

Referenced by `__polylog_exp_neg_int()`, `__polylog_exp_neg_real()`, `__polylog_exp_pos_int()`, and `__polylog_exp_↔pos_real()`.

9.3.2.35 `__clamp_pi()`

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__clamp_pi (
    std::complex< _Tp > __z )
```

Definition at line 171 of file `sf_polylog.tcc`.

Referenced by `__polylog_exp_neg_int()`, `__polylog_exp_neg_real()`, `__polylog_exp_pos_int()`, and `__polylog_exp_↔pos_real()`.

9.3.2.36 `__clausen()` [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__clausen (
    unsigned int __m,
    std::complex< _Tp > __z )
```

Return Clausen's function of integer order m and complex argument z . The notation and connection to polylog is from Wikipedia

Parameters

<code>↔ _m</code>	The non-negative integral order.
<code>↔ _z</code>	The complex argument.

Returns

The complex Clausen function.

Definition at line 1261 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

9.3.2.37 `__clausen()` [2/2]

```
template<typename _Tp >
_Tp std::__detail::__clausen (
    unsigned int __m,
    _Tp __x )
```

Return Clausen's function of integer order m and real argument x . The notation and connection to polylog is from Wikipedia

Parameters

<code>__m</code>	The integer order $m \geq 1$.
<code>__x</code>	The real argument.

Returns

The Clausen function.

Definition at line 1288 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

9.3.2.38 `__clausen_cl()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__clausen_cl (
    unsigned int __m,
    std::complex< _Tp > __z )
```

Return Clausen's cosine sum Cl_m for positive integer order m and complex argument w .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

\leftrightarrow _m	The integer order $m \geq 1$.
\leftrightarrow _z	The complex argument.

Returns

The Clausen cosine sum $Cl_m(w)$,

Definition at line 1372 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.39 `__clausen_cl()` [2/2]

```
template<typename _Tp >
_Tp std::__detail::__clausen_cl (
    unsigned int __m,
    _Tp __x )
```

Return Clausen's cosine sum Cl_m for positive integer order m and real argument w .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

\leftrightarrow _m	The integer order $m \geq 1$.
\leftrightarrow _x	The real argument.

Returns

The real Clausen cosine sum $Cl_m(w)$,

Definition at line 1400 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.40 __clausen_sl() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__clausen_sl (
    unsigned int __m,
    std::complex< _Tp > __z )
```

Return Clausen's sine sum Sl_m for positive integer order m and complex argument z .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

\leftrightarrow __m	The integer order $m \geq 1$.
\leftrightarrow __z	The complex argument.

Returns

The Clausen sine sum $Sl_m(w)$,

Definition at line 1316 of file sf_polylog.tcc.

References __polylog_exp().

9.3.2.41 __clausen_sl() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__clausen_sl (
    unsigned int __m,
    _Tp __x )
```

Return Clausen's sine sum Sl_m for positive integer order m and real argument x .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

\leftrightarrow __m	The integer order $m \geq 1$.
\leftrightarrow __x	The real argument.

Returns

The Clausen sine sum $Sl_m(w)$,

Definition at line 1344 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.42 `__comp_ellint_1()`

```
template<typename _Tp >
_Tp std::__detail::__comp_ellint_1 (
    _Tp __k )
```

Return the complete elliptic integral of the first kind $K(k)$ using the Carlson formulation.

The complete elliptic integral of the first kind is defined as

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

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

Parameters

<code>↔ __k</code>	The modulus of the complete elliptic function.
------------------------	--

Returns

The complete elliptic function of the first kind.

Definition at line 592 of file sf_ellint.tcc.

References `__comp_ellint_rf()`.

Referenced by `__ellint_1()`, `__ellnome_k()`, `__heuman_lambda()`, `__jacobi_zeta()`, `__theta_c()`, `__theta_d()`, `__theta_↔
n()`, and `__theta_s()`.

9.3.2.43 `__comp_ellint_2()`

```
template<typename _Tp >
_Tp std::__detail::__comp_ellint_2 (
    _Tp __k )
```

Return the complete elliptic integral of the second kind $E(k)$ using the Carlson formulation.

The complete elliptic integral of the second kind is defined as

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

Parameters

<code>__k</code>	The modulus of the complete elliptic function.
------------------	--

Returns

The complete elliptic function of the second kind.

Definition at line 666 of file `sf_ellint.tcc`.

References `__ellint_rd()`, and `__ellint_rf()`.

Referenced by `__ellint_2()`.

9.3.2.44 `__comp_ellint_3()`

```
template<typename _Tp >
_Tp std::__detail::__comp_ellint_3 (
    _Tp __k,
    _Tp __nu )
```

Return the complete elliptic integral of the third kind $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$ using the Carlson formulation.

The complete elliptic integral of the third kind is defined as

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

Parameters

<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 756 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

Referenced by `__ellint_3()`.

9.3.2.45 __comp_ellint_d()

```
template<typename _Tp >
_Tp std::__detail::__comp_ellint_d (
    _Tp __k )
```

Return the complete Legendre elliptic integral D.

Definition at line 862 of file sf_ellint.tcc.

References __ellint_rd().

9.3.2.46 __comp_ellint_rf()

```
template<typename _Tp >
_Tp std::__detail::__comp_ellint_rf (
    _Tp __x,
    _Tp __y )
```

Definition at line 252 of file sf_ellint.tcc.

Referenced by __comp_ellint_1(), and __ellint_rf().

9.3.2.47 __comp_ellint_rg()

```
template<typename _Tp >
_Tp std::__detail::__comp_ellint_rg (
    _Tp __x,
    _Tp __y )
```

Definition at line 368 of file sf_ellint.tcc.

Referenced by __ellint_rg().

9.3.2.48 __conf_hyperg()

```
template<typename _Tp >
_Tp std::__detail::__conf_hyperg (
    _Tp __a,
    _Tp __c,
    _Tp __x )
```

Return the confluent hypergeometric function ${}_1F_1(a; c; x) = M(a, c, x)$.

Parameters

\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 confluent hypergeometric function.

Definition at line 337 of file sf_hyperg.tcc.

References `__conf_hyperg_luke()`, `__conf_hyperg_series()`, and `__gnu_cxx::__fp_is_integer()`.

Referenced by `__tricomi_u_naive()`.

9.3.2.49 __conf_hyperg_lim()

```
template<typename _Tp >
_Tp std::__detail::__conf_hyperg_lim (
    _Tp __c,
    _Tp __x )
```

Return the confluent hypergeometric limit function ${}_0F_1(-; c; x)$.

Parameters

\leftrightarrow _c	The <i>denominator</i> parameter.
\leftrightarrow _x	The argument of the confluent hypergeometric limit function.

Returns

The confluent limit hypergeometric function.

Definition at line 163 of file sf_hyperg.tcc.

References `__conf_hyperg_lim_series()`, and `__gnu_cxx::__fp_is_integer()`.

9.3.2.50 __conf_hyperg_lim_series()

```
template<typename _Tp >
_Tp std::__detail::__conf_hyperg_lim_series (
    _Tp __c,
    _Tp __x )
```

This routine returns the confluent hypergeometric limit function by series expansion.

$${}_0F_1(-; c; x) = \Gamma(c) \sum_{n=0}^{\infty} \frac{1}{\Gamma(c+n)} \frac{x^n}{n!}$$

If a and b are integers and a < 0 and either b > 0 or b < a then the series is a polynomial with a finite number of terms.

Parameters

\leftrightarrow __c	The "denominator" parameter.
\leftrightarrow __x	The argument of the confluent hypergeometric limit function.

Returns

The confluent hypergeometric limit function.

Definition at line 130 of file sf_hyperg.tcc.

Referenced by __conf_hyperg_lim().

9.3.2.51 __conf_hyperg_luke()

```
template<typename _Tp >
_Tp std::__detail::__conf_hyperg_luke (
    _Tp __a,
    _Tp __c,
    _Tp __xin )
```

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

Like the case of the ${}_2F_1$ rational approximations, these are probably guaranteed to converge for $x < 0$, barring gross numerical instability in the pre-asymptotic regime.

Definition at line 231 of file sf_hyperg.tcc.

Referenced by __conf_hyperg().

9.3.2.52 `__conf_hyperg_series()`

```
template<typename _Tp >
_Tp std::__detail::__conf_hyperg_series (
    _Tp __a,
    _Tp __c,
    _Tp __x )
```

This routine returns the confluent hypergeometric function by series expansion.

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

Parameters

<code>__a</code>	The "numerator" parameter.
<code>__c</code>	The "denominator" parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 196 of file `sf_hyperg.tcc`.

Referenced by `__conf_hyperg()`.

9.3.2.53 `__cos_pi()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__cos_pi (
    _Tp __x )
```

Return the reperiodized cosine of argument x:

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

Definition at line 104 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()`.

9.3.2.54 __cos_pi() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cos_pi (
    std::complex< _Tp > __z )
```

Return the reperiodized cosine of complex argument z:

$$\cos_{\pi}(z) = \cos(\pi z) = \cos_{\pi}(x)\cosh_{\pi}(y) - i\sin_{\pi}(x)\sinh_{\pi}(y)$$

Definition at line 231 of file sf_trig.tcc.

References __cos_pi(), and __sin_pi().

9.3.2.55 __cosh_pi() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__cosh_pi (
    _Tp __x )
```

Return the reperiodized hyperbolic cosine of argument x:

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

Definition at line 133 of file sf_trig.tcc.

9.3.2.56 __cosh_pi() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cosh_pi (
    std::complex< _Tp > __z )
```

Return the reperiodized hyperbolic cosine of complex argument z:

$$\cosh_{\pi}(z) = \cosh_{\pi}(z) = \cosh_{\pi}(x)\cos_{\pi}(y) + i\sinh_{\pi}(x)\sin_{\pi}(y)$$

Definition at line 253 of file sf_trig.tcc.

References __cos_pi(), and __sin_pi().

9.3.2.57 __coshint()

```
template<typename _Tp >
_Tp std::__detail::__coshint (
    const _Tp __x )
```

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

The hyperbolic cosine integral is given by

$$Chi(x) = (Ei(x) - E_1(x))/2 = (Ei(x) + Ei(-x))/2$$

Parameters

\leftrightarrow	The argument of the hyperbolic cosine integral function.
x	

Returns

The hyperbolic cosine integral.

Definition at line 561 of file sf_expint.tcc.

References `__expint_E1()`, and `__expint_Ei()`.

9.3.2.58 __coulomb_CF1()

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__coulomb_CF1 (
    unsigned int __l,
    _Tp __eta,
    _Tp __x )
```

Evaluate the first continued fraction, giving the ratio F'/F at the upper l value. We also determine the sign of F at that point, since it is the sign of the last denominator in the continued fraction.

Definition at line 146 of file sf_coulomb.tcc.

9.3.2.59 __coulomb_CF2()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__coulomb_CF2 (
    unsigned int __l,
    _Tp __eta,
    _Tp __x )
```

Evaluate the second continued fraction to obtain the ratio

$$(G' + iF')/(G + iF) := P + iQ$$

at the specified l value.

Definition at line 204 of file sf_coulomb.tcc.

9.3.2.60 __coulomb_f_recur()

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__coulomb_f_recur (
    unsigned int __l_min,
    unsigned int __k_max,
    _Tp __eta,
    _Tp __x,
    _Tp _F_l_max,
    _Tp _Fp_l_max )
```

Evolve the backwards recurrence for F, F'.

$$F_{l-1} = (S_l F_l + F'_l) / R_l F'_{l-1} = (S_l F_{l-1} - R_l F_l)$$

where

$$R_l = \sqrt{1 + (\eta/l)^2} S_l = l/x + \eta/l$$

Definition at line 77 of file sf_coulomb.tcc.

9.3.2.61 __coulomb_g_recur()

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__coulomb_g_recur (
    unsigned int __l_min,
    unsigned int __k_max,
    _Tp __eta,
    _Tp __x,
    _Tp _G_l_min,
    _Tp _Gp_l_min )
```

Evolve the forward recurrence for G, G'.

$$G_{l+1} = (S_l G_l - G'_l) / R_l G'_{l+1} = R_{l+1} G_l - S_l G_{l+1}$$

where

$$R_l = \sqrt{1 + (\eta/l)^2} S_l = l/x + \eta/l$$

Definition at line 115 of file sf_coulomb.tcc.

9.3.2.62 __coulomb_norm()

```
template<typename _Tp >
_Tp std::__detail::__coulomb_norm (
    unsigned int __l,
    _Tp __eta )
```

Definition at line 49 of file sf_coulomb.tcc.

9.3.2.63 `__cyl_bessel()`

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cyl_bessel (
    std::complex< _Tp > __nu,
    std::complex< _Tp > __z )
```

Return the complex cylindrical Bessel function.

Parameters

in	<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 1173 of file `sf_hankel.tcc`.

References `__hankel()`.

9.3.2.64 `__cyl_bessel_i()`

```
template<typename _Tp >
_Tp std::__detail::__cyl_bessel_i (
    _Tp __nu,
    _Tp __x )
```

Return the regular modified Bessel function of order ν : $I_\nu(x)$.

The regular modified cylindrical Bessel function is:

$$I_\nu(x) = \sum_{k=0}^{\infty} \frac{(x/2)^{\nu+2k}}{k! \Gamma(\nu + k + 1)}$$

Parameters

<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 371 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_ik()`.

Referenced by `__rice_pdf()`.

9.3.2.65 `__cyl_bessel_ij_series()`

```
template<typename _Tp >
_Tp std::__detail::__cyl_bessel_ij_series (
    _Tp __nu,
    _Tp __x,
    _Tp __sgn,
    unsigned int __max_iter )
```

This routine returns the cylindrical Bessel functions of order ν : J_ν or I_ν by series expansion.

The modified cylindrical Bessel function is:

$$Z_\nu(x) = \sum_{k=0}^{\infty} \frac{\sigma^k (x/2)^{\nu+2k}}{k! \Gamma(\nu + k + 1)}$$

where $\sigma = +1$ or -1 for $Z = I$ or J respectively.

See Abramowitz & Stegun, 9.1.10 Abramowitz & Stegun, 9.6.7 (1) Handbook of Mathematical Functions, ed. Milton Abramowitz and Irene A. Stegun, Dover Publications, Equation 9.1.10 p. 360 and Equation 9.6.10 p. 375

Parameters

<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 434 of file sf_bessel.tcc.

References `__log_gamma()`.

Referenced by `__cyl_bessel_i()`, and `__cyl_bessel_j()`.

9.3.2.66 `__cyl_bessel_ik()`

```
template<typename _Tp >
__gnu_cxx::__cyl_mod_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_ik (
    _Tp __nu,
    _Tp __x )
```

Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.

Parameters

<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 309 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik_asymp()`, `__cyl_bessel_ik_stepped()`, and `__sin_pi()`.

Referenced by `__airy()`, `__cyl_bessel_i()`, `__cyl_bessel_k()`, and `__sph_bessel_ik()`.

9.3.2.67 `__cyl_bessel_ik_asymp()`

```
template<typename _Tp >
__gnu_cxx::__cyl_mod_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_ik_asymp (
    _Tp __nu,
    _Tp __x )
```

This routine computes the asymptotic modified cylindrical Bessel and functions of order 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_stepped()`.

9.3.2.68 `__cyl_bessel_ik_stepped()`

```
template<typename _Tp >
__gnu_cxx::__cyl_mod_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_ik_stepped (
    _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 153 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ik_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_ik()`.

9.3.2.69 `__cyl_bessel_j()`

```
template<typename _Tp >
_Tp std::__detail::__cyl_bessel_j (
    _Tp __nu,
    _Tp __x )
```

Return the Bessel function of order ν : $J_\nu(x)$.

The cylindrical Bessel function is:

$$J_\nu(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{\nu+2k}}{k! \Gamma(\nu + k + 1)}$$

Parameters

<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 581 of file `sf_bessel.tcc`.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_jn()`.

9.3.2.70 `__cyl_bessel_jn()`

```
template<typename _Tp >
__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn (
    _Tp __nu,
    _Tp __x )
```

Return the cylindrical Bessel functions and their derivatives of order ν by various means.

Definition at line 473 of file `sf_bessel.tcc`.

References `__cos_pi()`, `__cyl_bessel_jn_asymp()`, `__cyl_bessel_jn_steep()`, and `__sin_pi()`.

Referenced by `__airy()`, `__cyl_bessel_j()`, `__cyl_bessel_jn_neg_arg()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann_n()`, and `__sph_bessel_jn()`.

9.3.2.71 `__cyl_bessel_jn_asymp()`

```
template<typename _Tp >
__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn_asymp (
    _Tp __nu,
    _Tp __x )
```

This routine computes the asymptotic cylindrical Bessel and Neumann functions of order ν : $J_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.

$$J_\nu(z) = \left(\frac{2}{\pi z}\right)^{1/2} \left(\cos(\omega) \sum_{k=0}^{\infty} (-1)^k \frac{a_{2k}(\nu)}{z^{2k}} - \sin(\omega) \sum_{k=0}^{\infty} (-1)^k \frac{a_{2k+1}(\nu)}{z^{2k+1}} \right)$$

and

$$N_\nu(z) = \left(\frac{2}{\pi z}\right)^{1/2} \left(\sin(\omega) \sum_{k=0}^{\infty} (-1)^k \frac{a_{2k}(\nu)}{z^{2k}} + \cos(\omega) \sum_{k=0}^{\infty} (-1)^k \frac{a_{2k+1}(\nu)}{z^{2k+1}} \right)$$

where $\omega = z - \nu\pi/2 - \pi/4$ and

$$a_k(\nu) = \frac{(4\nu^2 - 1^2)(4\nu^2 - 3^2) \dots (4\nu^2 - (2k-1)^2)}{8^k k!}$$

These sums work everywhere but on the negative real axis: $|ph(z)| < \pi - \delta$.

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 100 of file `sf_bessel.tcc`.

Referenced by `__cyl_bessel_jn()`, and `__cyl_bessel_jn_steel()`.

9.3.2.72 `__cyl_bessel_jn_neg_arg()`

```
template<typename _Tp >
__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, std::complex<_Tp> > std::__detail::__cyl_bessel_jn_neg_arg (
    _Tp __nu,
    _Tp __x )
```

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

Definition at line 539 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()`.

9.3.2.73 `__cyl_bessel_jn_steel()`

```
template<typename _Tp >
__gnu_cxx::__cyl_bessel_t<_Tp, _Tp, _Tp> std::__detail::__cyl_bessel_jn_steel (
    _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 229 of file sf_bessel.tcc.

References `__cyl_bessel_jn_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_jn()`.

9.3.2.74 `__cyl_bessel_k()`

```
template<typename _Tp >
_Tp std::__detail::__cyl_bessel_k (
    _Tp __nu,
    _Tp __x )
```

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

The irregular modified Bessel function is defined by:

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

where for integral $\nu = n$ a limit is taken: $\lim_{\nu \rightarrow n}$. 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 405 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ik()`.

9.3.2.75 __cyl_hankel_1() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cyl_hankel_1 (
    _Tp __nu,
    _Tp __x )
```

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

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 638 of file sf_bessel.tcc.

References `__cyl_bessel_jn()`, `__cyl_bessel_jn_neg_arg()`, and `__polar_pi()`.

9.3.2.76 __cyl_hankel_1() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cyl_hankel_1 (
    std::complex< _Tp > __nu,
    std::complex< _Tp > __z )
```

Return the complex cylindrical Hankel function of the first kind.

Parameters

<code>in</code>	<code>__nu</code>	The order for which the cylindrical Hankel function of the first kind is evaluated.
<code>in</code>	<code>__z</code>	The argument at which the cylindrical Hankel function of the first kind is evaluated.

Returns

The complex cylindrical Hankel function of the first kind.

Definition at line 1139 of file sf_hankel.tcc.

References `__hankel()`.

9.3.2.77 `__cyl_hankel_2()` [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cyl_hankel_2 (
    _Tp __nu,
    _Tp __x )
```

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

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

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

Parameters

<code>__nu</code>	The order of the spherical Neumann function.
<code>__x</code>	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Definition at line 677 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`, `__cyl_bessel_jn_neg_arg()`, and `__polar_pi()`.

9.3.2.78 `__cyl_hankel_2()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cyl_hankel_2 (
    std::complex<_Tp> __nu,
    std::complex<_Tp> __z )
```

Return the complex cylindrical Hankel function of the second kind.

Parameters

in	<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 1156 of file sf_hankel.tcc.

References `__hankel()`.

9.3.2.79 __cyl_neumann()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__cyl_neumann (
    std::complex< _Tp > __nu,
    std::complex< _Tp > __z )
```

Return the complex cylindrical Neumann function.

Parameters

in	<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 1190 of file sf_hankel.tcc.

References `__hankel()`.

9.3.2.80 __cyl_neumann_n()

```
template<typename _Tp >
_Tp std::__detail::__cyl_neumann_n (
    _Tp __nu,
    _Tp __x )
```

Return the Neumann function of order ν : $N_\nu(x)$.

The Neumann function is defined by:

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

where for integral $\nu = n$ a limit is taken: $\lim_{\nu \rightarrow n}$.

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 612 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

9.3.2.81 `__dawson()`

```
template<typename _Tp >
_Tp std::__detail::__dawson (
    _Tp __x )
```

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

The Dawson integral is defined by:

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

and it's derivative is:

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

Parameters

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

Definition at line 235 of file `sf_dawson.tcc`.

References `__dawson_cont_frac()`, and `__dawson_series()`.

9.3.2.82 `__dawson_cont_frac()`

```
template<typename _Tp >
_Tp std::__detail::__dawson_cont_frac (
    _Tp __x )
```

Compute the Dawson integral using a sampling theorem representation.

This array could be built on a thread-local basis.

Definition at line 73 of file sf_dawson.tcc.

Referenced by __dawson().

9.3.2.83 __dawson_series()

```
template<typename _Tp >
_Tp std::__detail::__dawson_series (
    _Tp __x )
```

Compute the Dawson integral using the series expansion.

Definition at line 49 of file sf_dawson.tcc.

Referenced by __dawson().

9.3.2.84 __debye()

```
template<typename _Tp >
_Tp std::__detail::__debye (
    unsigned int __n,
    _Tp __x )
```

Return the Debye function. The Debye functions are related to the incomplete Riemann zeta function:

$$\zeta_x(s) = \frac{1}{\Gamma(s)} \int_0^x \frac{t^{s-1}}{e^t - 1} dt = \sum_{k=1}^{\infty} \frac{P(s, kx)}{k^s}$$

$$Z_x(s) = \frac{1}{\Gamma(s)} \int_x^{\infty} \frac{t^{s-1}}{e^t - 1} dt = \sum_{k=1}^{\infty} \frac{Q(s, kx)}{k^s}$$

where $P(a, x)$, $Q(a, x)$ is the incomplete gamma function ratios. The Debye function is:

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

Note the infinite limit:

$$D_n(\infty) = \int_0^{\infty} \frac{t^n}{e^t - 1} dt = n! \zeta(n+1)$$

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

Compute the Debye function:

$$D_n(x) = 1 - \sum_{k=1}^{\infty} e^{-kx} \frac{n}{k} \sum_{m=0}^n \frac{n!}{(n-m)!} \frac{1}{k^m} (kx)^m$$

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

Todo Find Debye for $x < -2\pi$!

Definition at line 916 of file `sf_zeta.tcc`.

9.3.2.85 `__debye_region()`

```
template<typename _Tp >
void std::__detail::__debye_region (
    std::complex< _Tp > __alpha,
    int & __indexr,
    char & __aorb )
```

Compute the Debye region in the complex plane.

Definition at line 53 of file `sf_hankel.tcc`.

Referenced by `__hankel()`.

9.3.2.86 `__digamma()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__digamma (
    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 3317 of file `sf_gamma.tcc`.

Referenced by `__digamma()`, `__hyperg_reflect()`, and `__polygamma()`.

9.3.2.87 __digamma() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__digamma (
    _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 3407 of file sf_gamma.tcc.

References `__digamma()`, `__digamma_asymp()`, `__gnu_cxx::__fp_is_half_odd_integer()`, `__gnu_cxx::__fp_is_integer()`, `__hurwitz_zeta()`, `std::__detail::_Factorial_table<_Tp>::__n`, and `__tan_pi()`.

9.3.2.88 __digamma_asymp()

```
template<typename _Tp >
_Tp std::__detail::__digamma_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 3374 of file sf_gamma.tcc.

Referenced by `__digamma()`.

9.3.2.89 __digamma_series()

```
template<typename _Tp >
_Tp std::__detail::__digamma_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 3342 of file sf_gamma.tcc.

9.3.2.90 __dilog()

```
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 246 of file sf_zeta.tcc.

9.3.2.91 __dirichlet_beta() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__dirichlet_beta (
    std::complex< _Tp > __s )
```

Return the Dirichlet beta function. Currently, s must be real (complex type but negligible imaginary part.) Otherwise `std::domain_error` is thrown. The Dirichlet beta function, in terms of the polylogarithm, is

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

Parameters

\leftrightarrow _s	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 1198 of file sf_polylog.tcc.

References `__polylog()`.

9.3.2.92 __dirichlet_beta() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__dirichlet_beta (
    _Tp __s )
```

Return the Dirichlet beta function for real argument. The Dirichlet beta function, in terms of the polylogarithm, is

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

Parameters

\leftrightarrow _s	The real argument.
-------------------------	--------------------

Returns

The Dirichlet Beta function of real argument.

Definition at line 1223 of file sf_polylog.tcc.

References `__polylog()`.

9.3.2.93 `__dirichlet_eta()` [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__dirichlet_eta (
    std::complex< _Tp > __s )
```

Return the Dirichlet eta function. Currently, s must be real (complex type but negligible imaginary part.) Otherwise `std::domain_error` is thrown. The Dirichlet eta function, in terms of the polylogarithm, is

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

Parameters

<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 1134 of file `sf_polylog.tcc`.

References `__polylog()`.

Referenced by `__dirichlet_eta()`, and `__dirichlet_lambda()`.

9.3.2.94 `__dirichlet_eta()` [2/2]

```
template<typename _Tp >
_Tp std::__detail::__dirichlet_eta (
    _Tp __s )
```

Return the Dirichlet eta function for real argument. The Dirichlet eta function, in terms of the polylogarithm, is

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

Parameters

<code>__s</code>	The real argument.
------------------	--------------------

Returns

The Dirichlet eta function.

Definition at line 1158 of file sf_polylog.tcc.

References `__dirichlet_eta()`, `__gnu_cxx::__fp_is_integer()`, `__gamma()`, `__polylog()`, and `__sin_pi()`.

9.3.2.95 __dirichlet_lambda()

```
template<typename _Tp >
_Tp std::__detail::__dirichlet_lambda (
    _Tp __s )
```

Return the Dirichlet lambda function for real argument.

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

Parameters

<code>__s</code>	The real argument.
------------------	--------------------

Returns

The Dirichlet lambda function.

Definition at line 1243 of file sf_polylog.tcc.

References `__dirichlet_eta()`, and `__riemann_zeta()`.

9.3.2.96 __double_factorial()

```
template<typename _Tp >
__GLIBCXX14_CONSTEXPR _Tp std::__detail::__double_factorial (
    int __n )
```

Return the double factorial of the integer n.

The double factorial is defined for integral n by:

$$n!! = 135\dots(n-2)n, \text{ odd } n!! = 246\dots(n-2)n, \text{ even } -1!! = 10!! = 1$$

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 1687 of file sf_gamma.tcc.

References `std::__detail::_Factorial_table<_Tp>::_factorial`, `__log_double_factorial()`, `std::__detail::_Factorial_table<_Tp>::_n`, `_S_double_factorial_table`, and `_S_neg_double_factorial_table`.

9.3.2.97 __ellint_1()

```
template<typename _Tp >
_Tp std::__detail::__ellint_1 (
    _Tp __k,
    _Tp __phi )
```

Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.

The incomplete elliptic integral of the first kind is defined as

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

Parameters

<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 621 of file sf_ellint.tcc.

References `__comp_ellint_1()`, and `__ellint_rf()`.

Referenced by `__heuman_lambda()`.

9.3.2.98 __ellint_2()

```
template<typename _Tp >
_Tp std::__detail::__ellint_2 (
    _Tp __k,
    _Tp __phi )
```

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

The incomplete elliptic integral of the second kind is defined as

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

Parameters

<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 702 of file sf_ellint.tcc.

References `__comp_ellint_2()`, `__ellint_rd()`, and `__ellint_rf()`.

9.3.2.99 __ellint_3()

```
template<typename _Tp >
_Tp std::__detail::__ellint_3 (
    _Tp __k,
    _Tp __nu,
    _Tp __phi )
```

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

The incomplete elliptic integral of the third kind is defined as

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

Parameters

<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 795 of file `sf_ellint.tcc`.

References `__comp_ellint_3()`, `__ellint_rf()`, and `__ellint_rj()`.

9.3.2.100 `__ellint_cel()`

```
template<typename _Tp >
_Tp std::__detail::__ellint_cel (
    _Tp __k_c,
    _Tp __p,
    _Tp __a,
    _Tp __b )
```

Return the Bulirsch complete elliptic integrals.

Definition at line 950 of file `sf_ellint.tcc`.

References `__ellint_rf()`, and `__ellint_rj()`.

9.3.2.101 `__ellint_d()`

```
template<typename _Tp >
_Tp std::__detail::__ellint_d (
    _Tp __k,
    _Tp __phi )
```

Return the Legendre elliptic integral D.

Definition at line 836 of file `sf_ellint.tcc`.

References `__ellint_rd()`.

9.3.2.102 `__ellint_el1()`

```
template<typename _Tp >
_Tp std::__detail::__ellint_el1 (
    _Tp __x,
    _Tp __k_c )
```

Return the Bulirsch elliptic integrals of the first kind.

Definition at line 878 of file `sf_ellint.tcc`.

References `__ellint_rf()`.

9.3.2.103 __ellint_el2()

```
template<typename _Tp >
_Tp std::__detail::__ellint_el2 (
    _Tp __x,
    _Tp __k_c,
    _Tp __a,
    _Tp __b )
```

Return the Bulirsch elliptic integrals of the second kind.

Definition at line 899 of file sf_ellint.tcc.

References `__ellint_rd()`, and `__ellint_rf()`.

9.3.2.104 __ellint_el3()

```
template<typename _Tp >
_Tp std::__detail::__ellint_el3 (
    _Tp __x,
    _Tp __k_c,
    _Tp __p )
```

Return the Bulirsch elliptic integrals of the third kind.

Definition at line 924 of file sf_ellint.tcc.

References `__ellint_rf()`, and `__ellint_rj()`.

9.3.2.105 __ellint_rc()

```
template<typename _Tp >
_Tp std::__detail::__ellint_rc (
    _Tp __x,
    _Tp __y )
```

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

The Carlson elliptic function is defined by:

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

Based on Carlson's algorithms:

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

Parameters

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

Returns

The Carlson elliptic function.

Definition at line 84 of file sf_ellint.tcc.

Referenced by __ellint_rf(), and __ellint_rj().

9.3.2.106 __ellint_rd()

```
template<typename _Tp >
_Tp std::__detail::__ellint_rd (
    _Tp __x,
    _Tp __y,
    _Tp __z )
```

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

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

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

Based on Carlson's algorithms:

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

Parameters

\leftrightarrow _x	The first of two symmetric arguments.
\leftrightarrow _y	The second of two symmetric arguments.
\leftrightarrow _z	The third argument.

Returns

The Carlson elliptic function of the second kind.

Definition at line 175 of file sf_ellint.tcc.

Referenced by __comp_ellint_2(), __comp_ellint_d(), __ellint_2(), __ellint_d(), __ellint_el2(), __ellint_rg(), and __ellint_rj().

9.3.2.107 __ellint_rf()

```
template<typename _Tp >
_Tp std::__detail::__ellint_rf (
    _Tp __x,
    _Tp __y,
    _Tp __z )
```

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

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

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

Parameters

$_x$	The first of three symmetric arguments.
$_y$	The second of three symmetric arguments.
$_z$	The third of three symmetric arguments.

Returns

The Carlson elliptic function of the first kind.

Definition at line 294 of file sf_ellint.tcc.

References __comp_ellint_rf(), and __ellint_rc().

Referenced by __comp_ellint_2(), __comp_ellint_3(), __ellint_1(), __ellint_2(), __ellint_3(), __ellint_cel(), __ellint_el1(), __ellint_el2(), __ellint_el3(), and __heuman_lambda().

9.3.2.108 `__ellint_rg()`

```
template<typename _Tp >
_Tp std::__detail::__ellint_rg (
    _Tp __x,
    _Tp __y,
    _Tp __z )
```

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

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

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

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.

Returns

The Carlson symmetric elliptic function of the second kind.

Definition at line 430 of file `sf_ellint.tcc`.

References `__comp_ellint_rg()`, and `__ellint_rd()`.

9.3.2.109 `__ellint_rj()`

```
template<typename _Tp >
_Tp std::__detail::__ellint_rj (
    _Tp __x,
    _Tp __y,
```



```

    _Tp __z,
    _Tp __p )

```

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

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

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

Based on Carlson's algorithms:

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

Parameters

<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.

Returns

The Carlson elliptic function of the fourth kind.

Definition at line 478 of file sf_ellint.tcc.

References `__ellint_rc()`, and `__ellint_rd()`.

Referenced by `__comp_ellint_3()`, `__ellint_3()`, `__ellint_cel()`, `__ellint_el3()`, `__heuman_lambda()`, and `__jacobi_zeta()`.

9.3.2.110 __ellnome()

```

template<typename _Tp >
_Tp std::__detail::__ellnome (
    _Tp __k )

```

Return the elliptic nome given the modulus k .

$$q(k) = \exp\left(-\pi \frac{K(k')}{K(k)}\right)$$

Definition at line 329 of file sf_theta.tcc.

References `__ellnome_k()`, and `__ellnome_series()`.

Referenced by `__theta_c()`, `__theta_d()`, `__theta_n()`, and `__theta_s()`.

9.3.2.111 `__ellnome_k()`

```
template<typename _Tp >
_Tp std::__detail::__ellnome_k (
    _Tp __k )
```

Use the arithmetic-geometric mean to calculate the elliptic nome given the elliptic argument k .

$$q(k) = \exp\left(-\pi \frac{K(k')}{K(k)}\right)$$

where $k' = \sqrt{1 - k^2}$ is the complementary elliptic argument and K is the Legendre elliptic integral of the first kind.

Definition at line 312 of file sf_theta.tcc.

References `__comp_ellint_1()`.

Referenced by `__ellnome()`.

9.3.2.112 `__ellnome_series()`

```
template<typename _Tp >
_Tp std::__detail::__ellnome_series (
    _Tp __k )
```

Use MacLaurin series to calculate the elliptic nome given the elliptic argument k .

$$q(k) = \exp\left(-\pi \frac{K(k')}{K(k)}\right)$$

where $k' = \sqrt{1 - k^2}$ is the complementary elliptic argument and K is the Legendre elliptic integral of the first kind.

Definition at line 291 of file sf_theta.tcc.

Referenced by `__ellnome()`.

9.3.2.113 `__euler()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__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 119 of file sf_euler.tcc.

9.3.2.114 `__euler()` [2/2]

```
template<typename _Tp >
_Tp std::__detail::__euler (
    unsigned int __n,
    _Tp __x )
```

Return the Euler polynomial $E_n(x)$ of order n at argument x.

The derivative is proportional to the previous polynomial:

$$E'_n(x) = nE_{n-1}(x)$$

$$E_n(1/2) = \frac{E_n}{2^n}, \text{ where } E_n \text{ is the n-th Euler number.}$$

Definition at line 137 of file sf_euler.tcc.

References `__bernoulli()`.

9.3.2.115 `__euler_series()`

```
template<typename _Tp >
_Tp std::__detail::__euler_series (
    unsigned int __n )
```

Return the Euler number from lookup or by series expansion.

The Euler numbers are given by the recursive sum:

$$E_n = B_n(1) = B_n$$

where $E_0 = 1$, $E_1 = 0$, $E_2 = -1$

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

Definition at line 61 of file sf_euler.tcc.

9.3.2.116 __eulerian_1()

```
template<typename _Tp >
_Tp std::__detail::__eulerian_1 (
    unsigned int __n,
    unsigned int __m ) [inline]
```

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

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

Note that $A(n, m)$ is a common older notation.

Definition at line 207 of file sf_euler.tcc.

9.3.2.117 __eulerian_1_recur()

```
template<typename _Tp >
_Tp std::__detail::__eulerian_1_recur (
    unsigned int __n,
    unsigned int __m )
```

Return the Eulerian number of the first kind. The Eulerian numbers of the first kind are defined by recursion:

$$\left\langle n \atop m \right\rangle = (n - m) \left\langle n - 1 \atop m - 1 \right\rangle + (m + 1) \left\langle n - 1 \atop m \right\rangle \text{ for } n > 0$$

Note that $A(n, m)$ is a common older notation.

Definition at line 166 of file sf_euler.tcc.

9.3.2.118 __eulerian_2()

```
template<typename _Tp >
_Tp std::__detail::__eulerian_2 (
    unsigned int __n,
    unsigned int __m ) [inline]
```

Return the Eulerian number of the second kind. The Eulerian numbers of the second kind are defined by recursion:

$$A(n, m) = (2n - m - 1)A(n - 1, m - 1) + (m + 1)A(n - 1, m) \text{ for } n > 0$$

Definition at line 254 of file sf_euler.tcc.

9.3.2.119 __eulerian_2_recur()

```
template<typename _Tp >
_Tp std::__detail::__eulerian_2_recur (
    unsigned int __n,
    unsigned int __m )
```

Return the Eulerian number of the second kind by recursion. The recursion is:

$$A(n, m) = (2n - m - 1)A(n - 1, m - 1) + (m + 1)A(n - 1, m) \text{ for } n > 0$$

Definition at line 219 of file sf_euler.tcc.

9.3.2.120 __exp2()

```
template<typename _Tp >
_Tp std::__detail::__exp2 (
    _Tp __x )
```

Make exp2 available to complex and real types.

Definition at line 64 of file sf_zeta.tcc.

Referenced by __riemann_zeta().

9.3.2.121 __expint() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__expint (
    unsigned int __n,
    _Tp __x )
```

Return the exponential integral $E_n(x)$.

The exponential integral is given by

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

Parameters

\leftrightarrow __n	The order of the exponential integral function.
\leftrightarrow __x	The argument of the exponential integral function.

Returns

The exponential integral.

Todo Study arbitrary switch to large- n $E_n(x)$.

Todo Find a good asymptotic switch point in $E_n(x)$.

Definition at line 476 of file sf_expint.tcc.

References `__expint_E1()`, `__expint_En_asymp()`, `__expint_En_cont_frac()`, `__expint_En_large_n()`, and `__expint_↔
En_series()`.

Referenced by `__logint()`.

9.3.2.122 __expint() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__expint (
    _Tp __x )
```

Return the exponential integral $Ei(x)$.

The exponential integral is given by

$$Ei(x) = - \int_{-x}^{\infty} \frac{e^t}{t} dt$$

Parameters

<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()`.

9.3.2.123 __expint_E1()



```
template<typename _Tp >
_Tp std::__detail::__expint_E1 (
    _Tp __x )
```

Return the exponential integral $E_1(x)$.

The exponential integral is given by

$$E_1(x) = \int_1^{\infty} \frac{e^{-xt}}{t} dt$$

Parameters

	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()`.

9.3.2.124 __expint_E1_asymp()


```
template<typename _Tp >
_Tp std::__detail::__expint_E1_asymp (
    _Tp __x )
```

Return the exponential integral $E_1(x)$ by asymptotic expansion.

The exponential integral is given by

$$E_1(x) = \int_1^{\infty} \frac{e^{-xt}}{t} dt$$

Parameters

	The argument of the exponential integral function.
<code>__x</code>	

Returns

The exponential integral.

Definition at line 114 of file sf_expint.tcc.

Referenced by `__expint_E1()`.

9.3.2.125 `__expint_E1_series()`


```
template<typename _Tp >
_Tp std::__detail::__expint_E1_series (
    _Tp __x )
```

Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.

The exponential integral is given by

$$E_1(x) = \int_1^{\infty} \frac{e^{-xt}}{t} dt$$

Parameters

	The argument of the exponential integral function.
<code>__x</code>	

Returns

The exponential integral.

Definition at line 76 of file sf_expint.tcc.

Referenced by `__expint_E1()`.

9.3.2.126 `__expint_Ei()`

```
template<typename _Tp >
_Tp std::__detail::__expint_Ei (
    _Tp __x )
```


Return the exponential integral $Ei(x)$.

The exponential integral is given by

$$Ei(x) = - \int_{-x}^{\infty} \frac{e^t}{t} dt$$

Parameters

\leftrightarrow _X	The argument of the exponential integral function.
-------------------------	--

Returns

The exponential integral.

Definition at line 356 of file sf_expint.tcc.

References `__expint_E1()`, `__expint_Ei_asyp()`, and `__expint_Ei_series()`.

Referenced by `__coshint()`, `__expint()`, `__expint_E1()`, and `__sinhint()`.

9.3.2.127 __expint_Ei_asyp()

```
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 _X	The argument of the exponential integral function.
-------------------------	--

Returns

The exponential integral.

Definition at line 322 of file sf_expint.tcc.

Referenced by `__expint_Ei()`.

9.3.2.128 `__expint_Ei_series()`

```
template<typename _Tp >
_Tp std::__detail::__expint_Ei_series (
    _Tp __x )
```

Return the exponential integral $Ei(x)$ by series summation.

The exponential integral is given by

$$Ei(x) = - \int_{-x}^{\infty} \frac{e^t}{t} dt$$

Parameters

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Returns

The exponential integral.

Definition at line 289 of file `sf_expint.tcc`.

Referenced by `__expint_Ei()`.

9.3.2.129 `__expint_En_asymp()`

```
template<typename _Tp >
_Tp std::__detail::__expint_En_asymp (
    unsigned int __n,
    _Tp __x )
```

Return the exponential integral $E_n(x)$ for large argument.

The exponential integral is given by

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

Parameters

<code>__n</code>	The order of the exponential integral function.
<code>__x</code>	The argument of the exponential integral function.

Returns

The exponential integral.

Definition at line 410 of file sf_expint.tcc.

Referenced by __expint().

9.3.2.130 __expint_En_cont_frac()

```
template<typename _Tp >
_Tp std::__detail::__expint_En_cont_frac (
    unsigned int __n,
    _Tp __x )
```

Return the exponential integral $E_n(x)$ by continued fractions.

The exponential integral is given by

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

Parameters

\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().

9.3.2.131 __expint_En_large_n()

```
template<typename _Tp >
_Tp std::__detail::__expint_En_large_n (
    unsigned int __n,
    _Tp __x )
```

Return the exponential integral $E_n(x)$ for large order.

The exponential integral is given by

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

Parameters

\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().

9.3.2.132 __expint_En_recursion()

```
template<typename _Tp >
_Tp std::__detail::__expint_En_recursion (
    unsigned int __n,
    _Tp __x )
```

Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.

The exponential integral is given by

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

Parameters

\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().

9.3.2.133 __expint_En_series()

```
template<typename _Tp >
_Tp std::__detail::__expint_En_series (
    unsigned int __n,
    _Tp __x )
```

Return the exponential integral $E_n(x)$ by series summation.

The exponential integral is given by

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

Parameters

\leftarrow __n	The order of the exponential integral function.
\leftarrow __x	The argument of the exponential integral function.

Returns

The exponential integral.

Definition at line 150 of file sf_expint.tcc.

Referenced by __expint().

9.3.2.134 __exponential_p()

```
template<typename _Tp >
_Tp std::__detail::__exponential_p (
    _Tp __lambda,
    _Tp __x )
```

Return the exponential cumulative probability density function.

The formula for the exponential cumulative probability density function is

$$F(x|\lambda) = 1 - e^{-\lambda x} \text{ for } x \geq 0$$

Definition at line 328 of file sf_distributions.tcc.

9.3.2.135 __exponential_pdf()

```
template<typename _Tp >
_Tp std::__detail::__exponential_pdf (
    _Tp __lambda,
    _Tp __x )
```

Return the exponential probability density function.

The formula for the exponential probability density function is

$$f(x|\lambda) = \lambda e^{-\lambda x} \text{ for } x \geq 0$$

Definition at line 308 of file sf_distributions.tcc.

9.3.2.136 __exponential_q()

```
template<typename _Tp >
_Tp std::__detail::__exponential_q (
    _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.

9.3.2.137 __factorial()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (
    unsigned int __n )
```

Return the factorial of the integer n.

The factorial is:

$$n! = 12...(n-1)n, 0! = 1$$

Definition at line 1617 of file sf_gamma.tcc.

References std::__detail::_Factorial_table<_Tp>::__n, and _S_factorial_table.

9.3.2.138 __falling_factorial() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__falling_factorial (
    _Tp __a,
    int __n )
```

Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and integral order n . The falling factorial function is defined by

$$a^{\underline{n}} = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $n^{\underline{n}} = n!$.

Definition at line 2941 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()`.

9.3.2.139 __falling_factorial() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__falling_factorial (
    _Tp __a,
    _Tp __nu )
```

Return the logarithm of the falling factorial function or the lower Pochhammer symbol for real argument a and order ν . The falling factorial function is defined by

$$a^{\underline{\nu}} = \Gamma(a + 1) / \Gamma(a - \nu + 1)$$

.

Definition at line 2996 of file sf_gamma.tcc.

References `__falling_factorial()`, `__gnu_cxx::__fp_is_integer()`, `__log_gamma()`, and `__log_gamma_sign()`.

9.3.2.140 __fermi_dirac()

```
template<typename _Sp , typename _Tp >
_Tp std::__detail::__fermi_dirac (
    _Sp __s,
    _Tp __x )
```

Return the Fermi-Dirac integral of integer or real order s and real argument x .

See also

https://en.wikipedia.org/wiki/Clausen_function
<http://dlmf.nist.gov/25.12.16>

$$F_s(x) = \frac{1}{\Gamma(s+1)} \int_0^\infty \frac{t^s}{e^{t-x} + 1} dt = -Li_{s+1}(-e^x)$$

Parameters

\leftrightarrow _s	The order $s > -1$.
\leftrightarrow _x	The real argument.

Returns

The real Fermi-Dirac integral $F_s(x)$,

Definition at line 1434 of file sf_polylog.tcc.

References `__polylog_exp()`.

9.3.2.141 __fisher_f_p()

```
template<typename _Tp >
_Tp std::__detail::__fisher_f_p (
    _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

__nu1	The number of degrees of freedom of sample 1
__nu2	The number of degrees of freedom of sample 2
__F	The F statistic

Definition at line 523 of file sf_distributions.tcc.

References `__beta_inc()`.

9.3.2.142 __fisher_f_pdf()

```
template<typename _Tp >
_Tp std::__detail::__fisher_f_pdf (
    _Tp __F,
    unsigned int __nu1,
    unsigned int __nu2 )
```

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .

The f-distribution propability function is related to the incomplete beta function:

$$Q(F|\nu_1, \nu_2) = I_{\frac{\nu_2}{\nu_2 + \nu_1 F}}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right)$$

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()`.

9.3.2.143 __fisher_f_q()

```
template<typename _Tp >
_Tp std::__detail::__fisher_f_q (
    _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()`.

9.3.2.144 `__fock_airy()`

```
template<typename _Tp >
__gnu_cxx::__fock_airy_t<_Tp, std::complex<_Tp> > std::__detail::__fock_airy (
    _Tp __x )
```

Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w'_1(x)$ and $w'_2(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$

.

Parameters

<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 560 of file sf_mod_bessel.tcc.

References `__airy()`.

9.3.2.145 `__fresnel()`

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__fresnel (
    const _Tp __x )
```

Return the Fresnel cosine and sine integrals as a complex number $C(x) + iS(x)$.

The Fresnel cosine integral is defined by:

$$C(x) = \int_0^x \cos\left(\frac{\pi}{2}t^2\right)dt$$

The Fresnel sine integral is defined by:

$$S(x) = \int_0^x \sin\left(\frac{\pi}{2}t^2\right)dt$$

Parameters

\leftrightarrow	The argument
x	

Definition at line 170 of file sf_fresnel.tcc.

References `__fresnel_cont_frac()`, and `__fresnel_series()`.

9.3.2.146 `__fresnel_cont_frac()`

```
template<typename _Tp >
void std::__detail::__fresnel_cont_frac (
    const _Tp __ax,
    _Tp & _Cf,
    _Tp & _Sf )
```

This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.

Definition at line 109 of file sf_fresnel.tcc.

Referenced by `__fresnel()`.

9.3.2.147 `__fresnel_series()`

```
template<typename _Tp >
void std::__detail::__fresnel_series (
    const _Tp __ax,
    _Tp & _Cf,
    _Tp & _Sf )
```

This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.

Definition at line 51 of file sf_fresnel.tcc.

Referenced by `__fresnel()`.

9.3.2.148 `__gamma()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__gamma (
    _Tp __a )
```

Return the gamma function $\Gamma(a)$. The gamma function is defined by:

$$\Gamma(a) = \int_0^{\infty} e^{-t} t^{a-1} dt (a > 0)$$

.

Parameters

\leftrightarrow _a	The argument of the gamma function.
-------------------------	-------------------------------------

Returns

The gamma function.

Definition at line 2639 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_p()`, `__gamma_pdf()`, `__gamma_q()`, \leftrightarrow `__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()`.

9.3.2.149 `__gamma()` [2/2]

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__gamma (
    _Tp __a,
    _Tp __x )
```

Return the incomplete gamma functions.

Definition at line 2766 of file sf_gamma.tcc.

References `__gnu_cxx::__fp_is_integer()`, `__gamma_cont_frac()`, and `__gamma_series()`.

9.3.2.150 `__gamma_cont_frac()`

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__gamma_cont_frac (
    _Tp __a,
    _Tp __x )
```

Return the incomplete gamma function by continued fraction.

Definition at line 2721 of file sf_gamma.tcc.

References `__log_gamma()`, `__log_gamma_sign()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__gamma()`, `__gamma_p()`, `__gamma_q()`, `__tgamma()`, and `__tgamma_lower()`.

9.3.2.151 __gamma_p() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__gamma_p (
    _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().

Referenced by __chi_squared_pdf().

9.3.2.152 __gamma_p() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__gamma_p (
    _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 2805 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma_cont_frac(), and __gamma_series().

9.3.2.153 `__gamma_pdf()`

```
template<typename _Tp >
_Tp std::__detail::__gamma_pdf (
    _Tp __alpha,
    _Tp __beta,
    _Tp __x )
```

Return the gamma propability distribution function.

The formula for the gamma probability density function is:

$$\Gamma(x|\alpha, \beta) = \frac{1}{\beta\Gamma(\alpha)} (x/\beta)^{\alpha-1} e^{-x/\beta}$$

Definition at line 121 of file `sf_distributions.tcc`.

References `__gamma()`.

9.3.2.154 `__gamma_q()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__gamma_q (
    _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()`.

Referenced by `__chi_squared_pdfc()`.

9.3.2.155 __gamma_q() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__gamma_q (
    _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 2839 of file sf_gamma.tcc.

References `__gnu_cxx::__fp_is_integer()`, `__gamma_cont_frac()`, and `__gamma_series()`.

9.3.2.156 __gamma_reciprocal()

```
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 2269 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()`.

9.3.2.157 `__gamma_reciprocal_series()`

```
template<typename _Tp >
_Tp std::__detail::__gamma_reciprocal_series (
    _Tp __a )
```

Return the reciprocal of the Gamma function by series. The reciprocal of the Gamma function is given by

$$\frac{1}{\Gamma(a)} = \sum_{k=1}^{\infty} c_k a^k$$

where the coefficients are defined by recursion:

$$c_{k+1} = \frac{1}{k} \left[\gamma_E c_k + (-1)^k \sum_{j=1}^{k-1} (-1)^j \zeta(j+1-k) c_j \right]$$

where $c_1 = 1$

Parameters

<code>__a</code>	The argument of the reciprocal of the gamma function.
------------------	---

Returns

The reciprocal of the gamma function.

Definition at line 2203 of file `sf_gamma.tcc`.

References `__gamma()`.

Referenced by `__gamma()`, `__gamma_reciprocal()`, and `__gamma_temme()`.

9.3.2.158 `__gamma_series()`

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__gamma_series (
    _Tp __a,
    _Tp __x )
```

Return the incomplete gamma function by series summation.

$$\gamma(a, x) = x^a e^{-x} \sum_{k=1}^{\infty} \frac{x^k}{(a)_k}$$

.

Definition at line 2676 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()`, `__gamma_p()`, `__gamma_q()`, `__tgamma()`, and `__tgamma_lower()`.

9.3.2.159 __gamma_temme()

```
template<typename _Tp >
__gnu_cxx::__gamma_temme_t<_Tp> std::__detail::__gamma_temme (
    _Tp __mu )
```

Compute the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ 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 188 of file sf_bessel.tcc.

References `__gamma_reciprocal_series()`.

Referenced by `__cyl_bessel_ik_steel()`, and `__cyl_bessel_jn_steel()`.

9.3.2.160 __gauss()

```
template<typename _Tp >
_Tp std::__detail::__gauss (
    _Tp __x )
```

The CDF of the normal distribution. i.e. the integrated lower tail of the normal PDF.

Definition at line 70 of file sf_owens_t.tcc.

9.3.2.161 `__gegenbauer_recur()`

```
template<typename _Tp >
__gnu_cxx::__gegenbauer_t<_Tp> std::__detail::__gegenbauer_recur (
    unsigned int __n,
    _Tp __lambda,
    _Tp __x )
```

Return the Gegenbauer polynomial $C_n^{(\lambda)}(x)$ of degree n and real order λ and argument x .

The Gegenbauer polynomials are generated by a three-term recursion relation:

$$C_n^{(\lambda)}(x) = \frac{1}{n} \left[2x(n + \lambda - 1)C_{n-1}^{(\lambda)}(x) - (n + 2\lambda - 2)C_{n-2}^{(\lambda)}(x) \right]$$

and $C_0^{(\lambda)}(x) = 1$, $C_1^{(\lambda)}(x) = 2\lambda x$.

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

<code>__n</code>	The non-negative integral degree
<code>__lambda</code>	The order
<code>__x</code>	The real argument

Definition at line 63 of file `sf_gegenbauer.tcc`.

9.3.2.162 `__gegenbauer_zeros()`

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__gegenbauer_zeros (
    unsigned int __n,
    _Tp __lambda )
```

Return a vector containing the zeros of the Gegenbauer or ultraspherical polynomial $C_n^{(\lambda)}$.

Definition at line 97 of file `sf_gegenbauer.tcc`.

References `__gnu_cxx::lgamma()`.

9.3.2.163 `__hankel()`

```
template<typename _Tp >
__gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>, std::complex<_Tp> > std::__detail::__hankel (
    std::complex< _Tp > __nu,
    std::complex< _Tp > __z )
```

Parameters

in	<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 1080 of file `sf_hankel.tcc`.

References `__debye_region()`, `__hankel_debye()`, and `__hankel_uniform()`.

Referenced by `__cyl_bessel()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann()`, and `__sph_hankel()`.

9.3.2.164 `__hankel_debye()`

```
template<typename _Tp >
__gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>, std::complex<_Tp> > std::__
detail::__hankel_debye (
    std::complex< _Tp > __nu,
    std::complex< _Tp > __z,
    std::complex< _Tp > __alpha,
    int __indexr,
    char & __aorb,
    int & __morn )
```

Parameters

in	<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 913 of file `sf_hankel.tcc`.

References `__sin_pi()`.

Referenced by `__hankel()`.

9.3.2.165 `__hankel_params()`

```
template<typename _Tp >
void std::__detail::__hankel_params (
    std::complex< _Tp > __nu,
    std::complex< _Tp > __zhat,
    std::complex< _Tp > & __p,
    std::complex< _Tp > & __p2,
    std::complex< _Tp > & __nup2,
    std::complex< _Tp > & __num2,
    std::complex< _Tp > & __num1d3,
    std::complex< _Tp > & __num2d3,
    std::complex< _Tp > & __num4d3,
    std::complex< _Tp > & __zeta,
    std::complex< _Tp > & __zetaphf,
    std::complex< _Tp > & __zetamhf,
    std::complex< _Tp > & __zetam3hf,
    std::complex< _Tp > & __zetrat )
```

Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.

Definition at line 108 of file `sf_hankel.tcc`.

Referenced by `__hankel_uniform_outer()`.

9.3.2.166 `__hankel_uniform()`

```
template<typename _Tp >
__gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>, std::complex<_Tp> > std::__↵
detail::__hankel_uniform (
    std::complex< _Tp > __nu,
    std::complex< _Tp > __z )
```

This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.

Parameters

in	<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 860 of file sf_hankel.tcc.

References `__hankel_uniform_olver()`.

Referenced by `__hankel()`.

9.3.2.167 `__hankel_uniform_olver()`

```
template<typename _Tp >
__gnu_cxx::__cyl_hankel_t<std::complex<_Tp>, std::complex<_Tp>, std::complex<_Tp> > std::__detail::__hankel_uniform_olver (
    std::complex< _Tp > __nu,
    std::complex< _Tp > __z )
```

Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order `nu` along with their derivatives.

Parameters

in	<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 777 of file sf_hankel.tcc.

References `__hankel_uniform_outer()`, and `__hankel_uniform_sum()`.

Referenced by `__hankel_uniform()`.

9.3.2.168 `__hankel_uniform_outer()`

```
template<typename _Tp >
void std::__detail::__hankel_uniform_outer (
    std::complex< _Tp > __nu,
    std::complex< _Tp > __z,
    _Tp __eps,
    std::complex< _Tp > & __zhat,
    std::complex< _Tp > & __ldnsq,
    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 ν 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 ν returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.

Definition at line 247 of file `sf_hankel.tcc`.

References `__airy_arg()`, and `__hankel_params()`.

Referenced by `__hankel_uniform_olver()`.

9.3.2.169 `__hankel_uniform_sum()`

```

template<typename _Tp >
void std::__detail::__hankel_uniform_sum (
    std::complex< _Tp > __p,
    std::complex< _Tp > __p2,
    std::complex< _Tp > __num2,
    std::complex< _Tp > __zetam3hf,
    std::complex< _Tp > _Aip,
    std::complex< _Tp > __o4dp,
    std::complex< _Tp > _Aim,
    std::complex< _Tp > __o4dm,
    std::complex< _Tp > __od2p,
    std::complex< _Tp > __od0dp,
    std::complex< _Tp > __od2m,
    std::complex< _Tp > __od0dm,
    _Tp __eps,
    std::complex< _Tp > & _H1sum,
    std::complex< _Tp > & _H1psum,
    std::complex< _Tp > & _H2sum,
    std::complex< _Tp > & _H2psum )

```

Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to `nterms` (less than 5) to achieve relative error `eps`.

Parameters

in	<code>__p</code>	
----	------------------	--

Parameters

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 324 of file `sf_hankel.tcc`.

Referenced by `__hankel_uniform_olver()`.

9.3.2.170 `__harmonic_number()`

```
template<typename _Tp >
_Tp std::__detail::__harmonic_number (
    unsigned int __n )
```

Definition at line 3286 of file `sf_gamma.tcc`.

References `std::__detail::_Factorial_table<_Tp>::__n`, `_S_harmonic_denom`, `_S_harmonic_numer`, and `_S_num_↵harmonic_numer`.

9.3.2.171 `__hermite()`

```
template<typename _Tp >
_Tp std::__detail::__hermite (
    unsigned int __n,
    _Tp __x )
```

This routine returns the Hermite polynomial of order n : $H_n(x)$.

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

An explicit series formula is:

$$H_n(x) = \sum_{k=0}^m \frac{(-1)^k}{k!(n-2k)!} (2x)^{n-2k} \text{ where } m = \left\lfloor \frac{n}{2} \right\rfloor$$

The Hermite polynomial obeys a reflection formula:

$$H_n(-x) = (-1)^n H_n(x)$$

Parameters

<code>__n</code>	The order of the Hermite polynomial.
<code>__x</code>	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x.

Definition at line 212 of file `sf_hermite.tcc`.

References `__hermite_asymp()`, and `__hermite_recur()`.

9.3.2.172 `__hermite_asymp()`

```
template<typename _Tp >
_Tp std::__detail::__hermite_asymp (
    unsigned int __n,
    _Tp __x )
```

This routine returns the Hermite polynomial of large order n: $H_n(x)$. We assume here that $x \geq 0$.

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 143 of file sf_hermite.tcc.

References `__airy()`.

Referenced by `__hermite()`.

9.3.2.173 `__hermite_recur()`

```
template<typename _Tp >
__gnu_cxx::__hermite_t<_Tp> std::__detail::__hermite_recur (
    unsigned int __n,
    _Tp __x )
```

This routine returns the Hermite polynomial of order n: $H_n(x)$ by recursion on n.

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

The Hermite polynomial has first and second derivatives:

$$H'_n(x) = 2nH_{n-1}(x)$$

and

$$H''_n(x) = 4n(n-1)H_{n-2}(x)$$

The Physicists Hermite polynomials have highest-order coefficient 2^n and are orthogonal with respect to the weight function

$$w(x) = e^{-x^2}$$

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 .

Todo Find the sign of Hermite blowup values.

Definition at line 86 of file `sf_hermite.tcc`.

Referenced by `__hermite()`.

9.3.2.174 __hermite_zeros()

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__hermite_zeros (
    unsigned int __n,
    _Tp __proto = _Tp{} )
```

Build a vector of the Gauss-Hermite integration rule abscissae and weights.

Definition at line 289 of file `sf_hermite.tcc`.

9.3.2.175 __heuman_lambda()

```
template<typename _Tp >
_Tp std::__detail::__heuman_lambda (
    _Tp __k,
    _Tp __phi )
```

Return the Heuman lambda function.

Definition at line 1008 of file `sf_ellint.tcc`.

References `__comp_ellint_1()`, `__ellint_1()`, `__ellint_rf()`, `__ellint_rj()`, and `__jacobi_zeta()`.

9.3.2.176 __hurwitz_zeta()

```
template<typename _Tp >
_Tp std::__detail::__hurwitz_zeta (
    _Tp __s,
    _Tp __a )
```

Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ 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

\leftrightarrow _s	The argument $s \neq 1$
\leftrightarrow _a	The scale parameter $a > -1$

Definition at line 871 of file sf_zeta.tcc.

References `__hurwitz_zeta_euler_maclaurin()`, and `__riemann_zeta()`.

Referenced by `__digamma()`, and `__polygamma()`.

9.3.2.177 `__hurwitz_zeta_euler_maclaurin()`

```
template<typename _Tp >
_Tp std::__detail::__hurwitz_zeta_euler_maclaurin (
    _Tp __s,
    _Tp __a )
```

Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ 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

\leftrightarrow _s	The argument $s \neq 1$
\leftrightarrow _a	The scale parameter $a > -1$

Definition at line 823 of file sf_zeta.tcc.

References `_S_Euler_Maclaurin_zeta`.

Referenced by `__hurwitz_zeta()`.

9.3.2.178 `__hurwitz_zeta_polylog()`

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__hurwitz_zeta_polylog (
```

```

_Tp __s,
std::complex< _Tp > __a )

```

Return the Hurwitz Zeta function for real s and complex a . This uses Jonquiere's identity:

$$\frac{(i2\pi)^s}{\Gamma(s)} \zeta(a, 1-s) = Li_s(e^{i2\pi a}) + (-1)^s Li_s(e^{-i2\pi a})$$

Parameters

\leftrightarrow __s	The real argument
\leftrightarrow __a	The complex parameter

Todo This `__hurwitz_zeta_polylog` prefactor is prone to overflow. positive integer orders s ?

Definition at line 1092 of file `sf_polylog.tcc`.

References `__gamma()`, and `__polylog_exp()`.

9.3.2.179 __hydrogen()

```

template<typename _Tp >
std::complex<_Tp> std::__detail::__hydrogen (
    unsigned int __n,
    unsigned int __l,
    unsigned int __m,
    _Tp __Z,
    _Tp __r,
    _Tp __theta,
    _Tp __phi )

```

Return the bound-state Coulomb wave-function.

Definition at line 248 of file `sf_coulomb.tcc`.

References `__assoc_laguerre()`, `__log_gamma()`, and `__sph_legendre()`.

9.3.2.180 __hyperg()

```
template<typename _Tp >
_Tp std::__detail::__hyperg (
    _Tp __a,
    _Tp __b,
    _Tp __c,
    _Tp __x )
```

Return the hypergeometric function ${}_2F_1(a, b; c; x)$.

The hypergeometric function is defined by

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

Parameters

\longleftrightarrow __a	The first <i>numerator</i> parameter.
\longleftrightarrow __b	The second <i>numerator</i> parameter.
\longleftrightarrow __c	The <i>denominator</i> parameter.
\longleftrightarrow __x	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 927 of file sf_hyperg.tcc.

References `__gnu_cxx::__fp_is_integer()`, `__hyperg_luke()`, `__hyperg_reflect()`, `__hyperg_series()`, `__log_gamma()`, and `__log_gamma_sign()`.

9.3.2.181 __hyperg_luke()

```
template<typename _Tp >
_Tp std::__detail::__hyperg_luke (
    _Tp __a,
    _Tp __b,
    _Tp __c,
    _Tp __xin )
```

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.

Definition at line 501 of file sf_hyperg.tcc.

Referenced by `__hyperg()`.

9.3.2.182 __hyperg_recur()

```
template<typename _Tp >
_Tp std::__detail::__hyperg_recur (
    int __m,
    _Tp __b,
    _Tp __c,
    _Tp __x )
```

Return the hypergeometric polynomial ${}_2F_1(-m, b; c; x)$ by Holm recursion.

The hypergeometric function is defined by

$${}_2F_1(-m, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(n-m)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

Parameters

\leftrightarrow __m	The first <i>numerator</i> parameter.
\leftrightarrow __b	The second <i>numerator</i> parameter.
\leftrightarrow __c	The <i>denominator</i> parameter.
\leftrightarrow __x	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

: go recur!

Definition at line 478 of file sf_hyperg.tcc.

9.3.2.183 __hyperg_reflect()

```
template<typename _Tp >
_Tp std::__detail::__hyperg_reflect (
    _Tp __a,
    _Tp __b,
    _Tp __c,
    _Tp __x )
```

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by the reflection formulae in Abramowitz & Stegun formula 15.3.6 for $d = c - a - b$ not integral and formula 15.3.11 for $d = c - a - b$ integral. This assumes $a, b, c \neq$ 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 637 of file sf_hyperg.tcc.

References `__digamma()`, `__gnu_cxx::__fp_is_integer()`, `__hyperg_series()`, `__log_gamma()`, and `__log_gamma_↵sign()`.

Referenced by `__hyperg()`.

9.3.2.184 __hyperg_series()

```
template<typename _Tp >
_Tp std::__detail::__hyperg_series (
    _Tp __a,
    _Tp __b,
    _Tp __c,
    _Tp __x )
```

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by series expansion.

The hypergeometric function is defined by

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

This works and it's pretty fast.

Parameters

<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 430 of file sf_hyperg.tcc.

Referenced by `__hyperg()`, and `__hyperg_reflect()`.

9.3.2.185 `__ibeta_cont_frac()`

```
template<typename _Tp >
_Tp std::__detail::__ibeta_cont_frac (
    _Tp __a,
    _Tp __b,
    _Tp __x )
```

Return the regularized incomplete beta function, $I_x(a, b)$, of arguments `a`, `b`, and `x`.

Parameters

<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()`.

9.3.2.186 `__jacobi_ellint()`

```
template<typename _Tp >
__gnu_cxx::__jacobi_ellint_t<_Tp> std::__detail::__jacobi_ellint (
    _Tp __k,
    _Tp __u )
```

Return a structure containing the three primary Jacobi elliptic functions: $sn(k, u)$, $cn(k, u)$, $dn(k, u)$.

Parameters

<code>__k</code>	The elliptic modulus $ k < 1$.
<code>__u</code>	The argument.

Returns

An object containing the three principal Jacobi elliptic functions, $sn(k, u)$, $cn(k, u)$, $dn(k, u)$ and the means to compute the remaining nine as well as the amplitude.

Definition at line 1648 of file sf_theta.tcc.

9.3.2.187 __jacobi_recur()

```
template<typename _Tp >
__gnu_cxx::__jacobi_t<_Tp> std::__detail::__jacobi_recur (
    unsigned int __n,
    _Tp __alpha1,
    _Tp __beta1,
    _Tp __x )
```

Compute the Jacobi polynomial by recursion on n :

$$2n(\alpha+\beta+n)(\alpha+\beta+2n-2)P_n^{(\alpha,\beta)}(x) = (\alpha+\beta+2n-1)((\alpha^2-\beta^2)+x(\alpha+\beta+2n-2)(\alpha+\beta+2n))P_{n-1}^{(\alpha,\beta)}(x) - 2(\alpha+n-1)(\beta+n-1)(\alpha+\beta+2n-2)P_{n-2}^{(\alpha,\beta)}(x)$$

Template Parameters

<code>_Tp</code>	The real type of the radial coordinate
------------------	--

Parameters

in	n	The order of the Jacobi polynomial
in	$\alpha1$	The first parameter of the Jacobi polynomial
in	$\beta1$	The second parameter of the Jacobi polynomial
in	x	The optional scaling of the coordinate; default 1.

Definition at line 66 of file sf_jacobi.tcc.

Referenced by `__radial_jacobi()`.

9.3.2.188 __jacobi_theta_1() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__jacobi_theta_1 (
    std::complex<_Tp > __q,
    std::complex<_Tp > __x )
```

Return the Jacobi θ_1 function by summation of the series.

The Jacobi or elliptic theta function is defined by

$$\theta_1(q, x) = 2 \sum_{n=1}^{\infty} (-1)^n q^{(n+\frac{1}{2})^2} \sin(2n+1)x$$

Regarding the nome and the theta function as functions of the lattice parameter $\tau = i \log(q)/\pi$ or $q = e^{i\pi\tau}$ the lattice parameter is transformed to maximize its imaginary part:

$$\theta_1(\tau + 1, x) = -ie^{i\pi/4} \theta_1(\tau, x)$$

and

$$\sqrt{-i\tau} \theta_1(\tau, x) = e^{(i\tau x^2/\pi)} \theta_1(\tau', \tau' x)$$

where the new lattice parameter is $\tau' = -1/\tau$.

The argument is reduced with

$$\theta_1(q, x + (m + n\tau)\pi) = (-1)^{m+n} q^{-n^2} e^{-2inx} \theta_1(q, x)$$

Parameters

\leftarrow _q	The elliptic nome, $ q < 1$.
\leftarrow _x	The argument.

Definition at line 979 of file sf_theta.tcc.

References `__jacobi_theta_1_prod()`, `__jacobi_theta_1_sum()`, `__polar_pi()`, `std::__detail::__jacobi_lattice_t< _Tp_ \leftarrow Omega1, _Tp_Omega3 >::__reduce()`, `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau()`, and `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__S_pi`.

Referenced by `__jacobi_theta_1()`.

9.3.2.189 `__jacobi_theta_1()` [2/2]

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_1 (
    _Tp _q,
    const _Tp _x )
```

Return the Jacobi θ_1 function for real nome and argument.

The Jacobi or elliptic theta function is defined by

$$\theta_1(q, x) = 2 \sum_{n=1}^{\infty} (-1)^n q^{(n+\frac{1}{2})^2} \sin(2n+1)x$$

Parameters

\leftrightarrow _q	The elliptic nome, $ q < 1$.
\leftrightarrow _x	The argument.

Definition at line 1047 of file sf_theta.tcc.

References `__jacobi_theta_1()`.

9.3.2.190 `__jacobi_theta_1_prod()`

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_1_prod (
    _Tp __q,
    _Tp __x )
```

Return the Jacobi θ_1 function by accumulation of the product.

The Jacobi or elliptic theta-1 function is defined by

$$\theta_1(q, x) = 2q^{1/4} \sin(x) \prod_{n=1}^{\infty} (1 - q^{2n})(1 - 2q^{2n} \cos(2x) + q^{4n})$$

Parameters

\leftrightarrow _q	The elliptic nome, $ q < 1$.
\leftrightarrow _x	The argument.

Definition at line 922 of file sf_theta.tcc.

Referenced by `__jacobi_theta_1()`.

9.3.2.191 `__jacobi_theta_1_sum()`

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_1_sum (
    _Tp __q,
    _Tp __x )
```

Return the Jacobi θ_1 function by summation of the series.

The Jacobi or elliptic theta-1 function is defined by

$$\theta_1(q, x) = 2 \sum_{n=1}^{\infty} (-1)^n q^{(n+\frac{1}{2})^2} \sin(2n+1)x$$

Parameters

\leftarrow _q	The elliptic nome, $ q < 1$.
\leftarrow _x	The argument.

Definition at line 887 of file sf_theta.tcc.

Referenced by __jacobi_theta_1().

9.3.2.192 __jacobi_theta_2() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__jacobi_theta_2 (
    std::complex< _Tp > __q,
    std::complex< _Tp > __x )
```

Return the Jacobi θ_2 function by summation of the series.

The Jacobi or elliptic theta function is defined by

$$\theta_2(q, x) = 2 \sum_{n=1}^{\infty} q^{(n+\frac{1}{2})^2} \cos(2n+1)x$$

Regarding the nome and the theta function as functions of the lattice parameter $\tau = i \log(q)/\pi$ or $q = e^{i\pi\tau}$ the lattice parameter is transformed to maximize its imaginary part:

$$\theta_2(\tau + 1, x) = e^{i\pi/4} \theta_2(\tau, x)$$

and

$$\sqrt{-i\tau} \theta_2(\tau, x) = e^{(i\tau x^2/\pi)} \theta_4(\tau', \tau' x)$$

where the new lattice parameter is $\tau' = -1/\tau$.

The argument is reduced with

$$\theta_2(q, x + (m + n\tau)\pi) = (-1)^m q^{-n^2} e^{-2inx} \theta_2(q, x)$$

Parameters

\leftarrow _q	The elliptic nome, $ q < 1$.
\leftarrow _x	The argument.

Definition at line 1175 of file sf_theta.tcc.

References __jacobi_theta_2_prod(), __jacobi_theta_2_sum(), __jacobi_theta_4_sum(), __polar_pi(), std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__reduce(), std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau(), std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__S_pi, and std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th2.

Referenced by __jacobi_theta_2().

9.3.2.193 __jacobi_theta_2() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_2 (
    _Tp __q,
    const _Tp __x )
```

Return the Jacobi θ_2 function for real nome and argument.

The Jacobi or elliptic theta function is defined by

$$\theta_2(q, x) = 2 \sum_{n=1}^{\infty} q^{(n+\frac{1}{2})^2} \cos(2n+1)x$$

Parameters

\leftarrow _q	The elliptic nome, $ q < 1$.
\leftarrow _x	The argument.

Definition at line 1248 of file sf_theta.tcc.

References __jacobi_theta_2().

9.3.2.194 __jacobi_theta_2_prod()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_2_prod (
    _Tp __q,
    _Tp __x )
```

Return the Jacobi θ_2 function by accumulation of the product.

The Jacobi or elliptic theta-2 function is defined by

$$\theta_2(q, x) = 2q^{1/4} \sin(x) \prod_{n=1}^{\infty} (1 - q^{2n})(1 + 2q^{2n} \cos(2x) + q^{4n})$$

Parameters

\hookleftarrow __q	The elliptic nome, $ q < 1$.
\hookleftarrow __x	The argument.

Definition at line 1108 of file sf_theta.tcc.

References __jacobi_theta_4_prod(), and __jacobi_theta_4_sum().

Referenced by __jacobi_theta_2().

9.3.2.195 __jacobi_theta_2_sum()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_2_sum (
    _Tp __q,
    _Tp __x )
```

Return the Jacobi θ_2 function by summation of the series.

The Jacobi or elliptic theta-2 function is defined by

$$\theta_2(q, x) = 2 \sum_{n=1}^{\infty} q^{(n+\frac{1}{2})^2} \cos(2n+1)x$$

Parameters

\hookleftarrow __q	The elliptic nome, $ q < 1$.
\hookleftarrow __x	The argument.

Definition at line 1076 of file sf_theta.tcc.

Referenced by __jacobi_theta_2(), and __jacobi_theta_4().

9.3.2.196 __jacobi_theta_3() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__jacobi_theta_3 (
    std::complex< _Tp > __q,
    std::complex< _Tp > __x )
```

Return the Jacobi θ_3 function by summation of the series.

The Jacobi or elliptic theta function is defined by

$$\theta_3(q, x) = 1 + 2 \sum_{n=1}^{\infty} q^{n^2} \cos 2nx$$

Regarding the nome and the theta function as functions of the lattice parameter $\tau = i \log(q)/\pi$ or $q = e^{i\pi\tau}$ the lattice parameter is transformed to maximize its imaginary part:

$$\theta_3(\tau + 1, x) = \theta_3(\tau, x)$$

and

$$\sqrt{-i\tau} \theta_3(\tau, x) = e^{(i\tau x^2/\pi)} \theta_3(\tau', \tau' x)$$

where the new lattice parameter is $\tau' = -1/\tau$.

The argument is reduced with

$$\theta_3(q, x + (m + n\tau)\pi) = q^{-n^2} e^{-2inx} \theta_3(q, x)$$

Parameters

\leftarrow __q	The elliptic nome, $ q < 1$.
\leftarrow __x	The argument.

Definition at line 1364 of file sf_theta.tcc.

References __jacobi_theta_3_prod(), __jacobi_theta_3_sum(), std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__reduce(), std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau(), std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__S_pi, and std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th3.

Referenced by __jacobi_theta_3().

9.3.2.197 `__jacobi_theta_3()` [2/2]

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_3 (
    _Tp __q,
    const _Tp __x )
```

Return the Jacobi θ_3 function for real nome and argument.

The Jacobi or elliptic theta function is defined by

$$\theta_3(q, x) = 1 + 2 \sum_{n=1}^{\infty} q^{n^2} \cos 2nx$$

Parameters

<code>__q</code>	The elliptic nome, $ q < 1$.
<code>__x</code>	The argument.

Definition at line 1432 of file `sf_theta.tcc`.

References `__jacobi_theta_3()`.

9.3.2.198 `__jacobi_theta_3_prod()`

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_3_prod (
    _Tp __q,
    _Tp __x )
```

Return the Jacobi θ_3 function by accumulation of the product.

The Jacobi or elliptic theta-3 function is defined by

$$\theta_3(q, x) = \prod_{n=1}^{\infty} (1 - q^{2n})(1 + 2q^{2n-1} \cos(2x) + q^{4n-2})$$

Parameters

<code>__q</code>	The elliptic nome, $ q < 1$.
<code>__x</code>	The argument.

Definition at line 1308 of file sf_theta.tcc.

Referenced by __jacobi_theta_3().

9.3.2.199 __jacobi_theta_3_sum()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_3_sum (
    _Tp __q,
    _Tp __x )
```

Return the Jacobi θ_3 function by summation of the series.

The Jacobi or elliptic theta-3 function is defined by

$$\theta_3(q, x) = 1 + 2 \sum_{n=1}^{\infty} q^{n^2} \cos 2nx$$

Parameters

\leftarrow __q	The elliptic nome, $ q < 1$.
\leftarrow __x	The argument.

Definition at line 1276 of file sf_theta.tcc.

Referenced by __jacobi_theta_3().

9.3.2.200 __jacobi_theta_4() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__jacobi_theta_4 (
    std::complex<_Tp> __q,
    std::complex<_Tp> __x )
```

Return the Jacobi θ_4 function by summation of the series.

The Jacobi or elliptic theta-4 function is defined by

$$\theta_4(q, x) = 1 + 2 \sum_{n=1}^{\infty} (-1)^n q^{n^2} \cos 2nx$$

Regarding the nome and the theta function as functions of the lattice parameter $\tau = i\log(q)/\pi$ or $q = e^{i\pi\tau}$ the lattice parameter is transformed to maximize its imaginary part:

$$\theta_4(\tau + 1, x) = \theta_4(\tau, x)$$

and

$$\sqrt{-i\tau}\theta_4(\tau, x) = e^{(i\tau x^2/\pi)}\theta_2(\tau', \tau'x)$$

where the new lattice parameter is $\tau' = -1/\tau$.

The argument is reduced with

$$\theta_4(q, z + (m + n\tau)\pi) = (-1)^n q^{-n^2} e^{-2inz} \theta_4(q, z)$$

Parameters

\leftarrow _q	The elliptic nome, $ q < 1$.
\leftarrow _x	The argument.

Definition at line 1550 of file sf_theta.tcc.

References `__jacobi_theta_2_sum()`, `__jacobi_theta_4_prod()`, `__jacobi_theta_4_sum()`, `std::__detail::__jacobi_lattice_t<_Tp_Omega1, _Tp_Omega3>::__reduce()`, `std::__detail::__jacobi_lattice_t<_Tp_Omega1, _Tp_Omega3>::__tau()`, `std::__detail::__jacobi_lattice_t<_Tp_Omega1, _Tp_Omega3>::__S_pi`, and `std::__detail::__jacobi_theta_0_t<_Tp1, _Tp3>::th4`.

Referenced by `__jacobi_theta_4()`.

9.3.2.201 __jacobi_theta_4() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_4 (
    _Tp __q,
    const _Tp __x )
```

Return the Jacobi θ_4 function for real nome and argument.

The Jacobi or elliptic theta function is defined by

$$\theta_4(q, x) = 1 + 2 \sum_{n=1}^{\infty} (-1)^n q^{n^2} \cos 2nx$$

Parameters

\leftarrow _q	The elliptic nome, $ q < 1$.
\leftarrow _x	The argument.

Definition at line 1621 of file sf_theta.tcc.

References `__jacobi_theta_4()`.

9.3.2.202 `__jacobi_theta_4_prod()`

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_4_prod (
    _Tp __q,
    _Tp __x )
```

Return the Jacobi θ_4 function by accumulation of the product.

The Jacobi or elliptic theta-4 function is defined by

$$\theta_4(q, x) = \prod_{n=1}^{\infty} (1 - q^{2n})(1 - 2q^{2n-1} \cos(2x) + q^{4n-2})$$

Parameters

\leftrightarrow <code>__q</code>	The elliptic nome, $ q < 1$.
\leftrightarrow <code>__x</code>	The argument.

Definition at line 1494 of file sf_theta.tcc.

Referenced by `__jacobi_theta_2_prod()`, and `__jacobi_theta_4()`.

9.3.2.203 `__jacobi_theta_4_sum()`

```
template<typename _Tp >
_Tp std::__detail::__jacobi_theta_4_sum (
    _Tp __q,
    _Tp __x )
```

Return the Jacobi θ_4 function by summation of the series.

The Jacobi or elliptic theta function is defined by

$$\theta_4(q, x) = 1 + 2 \sum_{n=1}^{\infty} (-1)^n q^{n^2} \cos 2nx$$

Parameters

\leftrightarrow _q	The elliptic nome, $ q < 1$.
\leftrightarrow _x	The argument.

Definition at line 1460 of file sf_theta.tcc.

Referenced by __jacobi_theta_2(), __jacobi_theta_2_prod(), and __jacobi_theta_4().

9.3.2.204 __jacobi_zeros()

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__jacobi_zeros (
    unsigned int __n,
    _Tp __alpha1,
    _Tp __beta1 )
```

Return a vector containing the zeros of the Jacobi polynomial $P_n^{(\alpha,\beta)}(x)$.

Template Parameters

_Tp	The real type of the radial coordinate
-----	--

Parameters

in	<i>n</i>	The order of the Jacobi polynomial
in	<i>alpha1</i>	The first parameter of the Jacobi polynomial
in	<i>beta1</i>	The second parameter of the Jacobi polynomial

Definition at line 139 of file sf_jacobi.tcc.

References __gnu_cxx::lgamma().

Referenced by __radial_jacobi_zeros().

9.3.2.205 __jacobi_zeta()

```
template<typename _Tp >
_Tp std::__detail::__jacobi_zeta (
```

```

    _Tp __k,
    _Tp __phi )

```

Return the Jacobi zeta function.

Definition at line 971 of file sf_ellint.tcc.

References `__comp_ellint_1()`, and `__ellint_rj()`.

Referenced by `__heuman_lambda()`.

9.3.2.206 __kolmogorov_p()

```

template<typename _Tp >
_Tp std::__detail::__kolmogorov_p (
    _Tp __a,
    _Tp __b,
    _Tp __x )

```

$$P(K \leq x) = 1 - e^{-2x^2} + e^{-2 \cdot 4x^2} + e^{-2 \cdot 9x^2} - e^{-2 \cdot 16x^2} + \dots$$

Definition at line 723 of file sf_distributions.tcc.

9.3.2.207 __laguerre() [1/2]

```

template<typename _Tpa , typename _Tp >
_Tp std::__detail::__laguerre (
    unsigned int __n,
    _Tpa __alpha1,
    _Tp __x )

```

This routine returns the associated Laguerre polynomial of degree n, order 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 order $\alpha = m$ 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 degree of the Laguerre function.
<code>__alpha1</code>	The order of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

Returns

The value of the Laguerre function of degree n , order α , and argument x .

Definition at line 316 of file `sf_laguerre.tcc`.

References `__laguerre_hyperg()`, `__laguerre_large_n()`, and `__laguerre_recur()`.

9.3.2.208 `__laguerre()` [2/2]

```
template<typename _Tp >
_Tp std::__detail::__laguerre (
    unsigned int __n,
    _Tp __x )
```

This routine returns the Laguerre polynomial of degree n : $L_n(x)$.

The Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})$$

Parameters

<code>↔ __n</code>	The degree of the Laguerre polynomial.
<code>↔ __x</code>	The argument of the Laguerre polynomial.

Returns

The value of the Laguerre polynomial of order n and argument x .

Definition at line 386 of file `sf_laguerre.tcc`.

9.3.2.209 __laguerre_hyperg()

```
template<typename _Tpa , typename _Tp >
_Tp std::__detail::__laguerre_hyperg (
    unsigned int __n,
    _Tpa __alpha1,
    _Tp __x )
```

Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.

The associated Laguerre function is defined by

$$L_n^{(\alpha)}(x) = \frac{(\alpha + 1)_n}{n!} {}_1F_1(-n; \alpha + 1; x)$$

where $(\alpha)_n$ is the Pochhammer symbol and ${}_1F_1(a; c; x)$ is the confluent hypergeometric function.

This function assumes $x \neq 0$.

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 degree of the Laguerre function.
<code>__alpha1</code>	The order of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

Returns

The value of the Laguerre function of degree n , order α , and argument x .

Definition at line 131 of file `sf_laguerre.tcc`.

Referenced by `__laguerre()`.

9.3.2.210 __laguerre_large_n()

```
template<typename _Tpa , typename _Tp >
_Tp std::__detail::__laguerre_large_n (
    unsigned __n,
    _Tpa __alpha1,
    _Tp __x )
```

This routine returns the associated Laguerre polynomial of degree n , order $\alpha > -1$ 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 degree of the Laguerre function.
<code>__alpha1</code>	The order of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

Returns

The value of the Laguerre function of degree n , order α , and argument x .

This is from the GNU Scientific Library.

Definition at line 75 of file `sf_laguerre.tcc`.

References `__log_gamma()`, and `__sin_pi()`.

Referenced by `__laguerre()`.

9.3.2.211 `__laguerre_recur()`

```
template<typename _Tpa , typename _Tp >
__gnu_cxx::__laguerre_t<_Tpa, _Tp> std::__detail::__laguerre_recur (
    unsigned int __n,
    _Tpa __alpha1,
    _Tp __x )
```

This routine returns the associated Laguerre polynomial of degree n , order 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 order $\alpha = m$ 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 degree of the Laguerre function.
<code>__alpha1</code>	The order 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 189 of file `sf_laguerre.tcc`.

Referenced by `__laguerre()`.

9.3.2.212 `__laguerre_zeros()`

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__laguerre_zeros (
    unsigned int __n,
    _Tp __alpha1 )
```

Return an array of abscissae and weights for the Gauss-Laguerre rule.

Definition at line 225 of file `sf_laguerre.tcc`.

References `__gnu_cxx::lgamma()`.

9.3.2.213 `__lanczos_binet1p()`

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_binet1p (
    _Tp __z )
```

Return the Binet function $J(1+z)$ by the Lanczos method. The Binet function is the log of the scaled Gamma function $\log(\Gamma^*(z))$ defined by

$$J(z) = \log(\Gamma^*(z)) = \log(\Gamma(z)) + z - \left(z - \frac{1}{2}\right) \log(z) - \log(2\pi)$$

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>__z</code>	The argument of the log of the gamma function.
------------------	--

Returns

The logarithm of the gamma function.

Definition at line 2125 of file `sf_gamma.tcc`.

References `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__lanczos_log_gamma1p()`.

9.3.2.214 __lanczos_log_gamma1p()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_log_gamma1p (
    _Tp __z )
```

Return the logarithm of the gamma function $\log(\Gamma(1 + z))$ by the Lanczos method.

If the argument is real, the log of the absolute value of the Gamma function is returned. The sign to be applied to the exponential of this log Gamma can be recovered with a call to `__log_gamma_sign`.

For complex argument the fully complex log of the gamma function is returned.

Parameters

<code>__z</code>	The argument of the log of the gamma function.
------------------	--

Returns

The logarithm of the gamma function.

Definition at line 2159 of file `sf_gamma.tcc`.

References `__lanczos_binet1p()`, and `__sin_pi()`.

9.3.2.215 __legendre_p()

```
template<typename _Tp >
__gnu_cxx::__legendre_p_t<_Tp> std::__detail::__legendre_p (
    unsigned int __l,
    _Tp __x )
```

Return the Legendre polynomial by upward recursion on degree l .

The Legendre function of degree 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

\leftarrow __l	The degree of the Legendre polynomial. $l \geq 0$.
\leftarrow __x	The argument of the Legendre polynomial.

Definition at line 82 of file sf_legendre.tcc.

Referenced by __assoc_legendre_p(), and __sph_legendre().

9.3.2.216 __legendre_q()

```
template<typename _Tp >
_Tp std::__detail::__legendre_q (
    unsigned int __l,
    _Tp __x )
```

Return the Legendre function of the second kind by upward recursion on degree l .

The Legendre function of the second kind of degree 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

\leftrightarrow _l	The degree of the Legendre function. $l \geq 0$.
\leftrightarrow _x	The argument of the Legendre function. $ x \leq 1$.

Definition at line 141 of file sf_legendre.tcc.

9.3.2.217 __legendre_zeros()

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__legendre_zeros (
    unsigned int __l,
    _Tp proto = _Tp{} )
```

Build a list of zeros and weights for the Gauss-Legendre integration rule for the Legendre polynomial of degree l .

Definition at line 390 of file sf_legendre.tcc.

9.3.2.218 __log_binomial() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__log_binomial (
    unsigned int __n,
    unsigned int __k )
```

Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

The binomial coefficients are generated by:

$$(1+t)^n = \sum_{k=0}^n \binom{n}{k} t^k$$

.

Parameters

\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 2434 of file sf_gamma.tcc.

References `__log_gamma()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__binomial()`.

9.3.2.219 __log_binomial() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__log_binomial (
    _Tp __nu,
    unsigned int __k )
```

Return the logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu + 1)}{\Gamma(\nu - k + 1)\Gamma(k + 1)}$$

The binomial coefficients are generated by:

$$(1 + t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

Parameters

<code>__nu</code>	The first argument of the binomial coefficient.
<code>__k</code>	The second argument of the binomial coefficient.

Returns

The logarithm of the binomial coefficient.

Definition at line 2471 of file sf_gamma.tcc.

References `__log_gamma()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

9.3.2.220 __log_binomial_sign() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__log_binomial_sign (
```

```

    _Tp __nu,
    unsigned int __k )

```

Return the sign of the exponentiated logarithm of the binomial coefficient for non-integral degree. The binomial coefficient is given by:

$$\binom{\nu}{k} = \frac{\Gamma(\nu + 1)}{\Gamma(\nu - k + 1)\Gamma(k + 1)}$$

The binomial coefficients are generated by:

$$(1 + t)^\nu = \sum_{k=0}^{\infty} \binom{\nu}{k} t^k$$

Parameters

<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 2502 of file `sf_gamma.tcc`.

References `__log_gamma_sign()`, and `std::__detail::_Factorial_table<_Tp>::__n`.

Referenced by `__binomial()`.

9.3.2.221 `__log_binomial_sign()` [2/2]

```

template<typename _Tp >
std::complex<_Tp> std::__detail::__log_binomial_sign (
    std::complex<_Tp> __nu,
    unsigned int __k )

```

Definition at line 2517 of file `sf_gamma.tcc`.

9.3.2.222 `__log_double_factorial()` [1/2]

```

template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (
    _Tp __nu )

```

Extend double factorial to non-integer arguments. Arkken,

$$\log(\nu!!) = \frac{\nu}{2} \log(2) + (\cos(\pi\nu) - 1) \log(\pi/2)/4 + \log(\Gamma(1 + \nu/2))$$

Definition at line 1657 of file `sf_gamma.tcc`.

References `__cos_pi()`, and `__log_gamma()`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

9.3.2.223 __log_double_factorial() [2/2]

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (
    int __n )
```

Return the logarithm of the double factorial of the integer n.

The double factorial is defined for integral n by:

$$n!! = 135...(n-2)n, \text{ odd } n!! = 246...(n-2)n, \text{ even } -1!! = 10!! = 1$$

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 1727 of file sf_gamma.tcc.

References `__log_double_factorial()`, `std::__detail::_Factorial_table<_Tp>::__log_factorial`, `std::__detail::_Factorial_table<_Tp>::__n`, `_S_double_factorial_table`, and `_S_neg_double_factorial_table`.

9.3.2.224 __log_factorial()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_factorial (
    unsigned int __n )
```

Return the logarithm of the factorial of the integer n.

The factorial is:

$$n! = 12...(n-1)n, 0! = 1$$

Definition at line 1635 of file sf_gamma.tcc.

References `__log_gamma()`, `std::__detail::_Factorial_table<_Tp>::__n`, `_S_double_factorial_table`, and `_S_factorial_table`.

9.3.2.225 `__log_falling_factorial()`

```
template<typename _Tp >
_Tp std::__detail::__log_falling_factorial (
    _Tp __a,
    _Tp __nu )
```

Return the logarithm of the falling factorial function or the lower Pochhammer symbol. The lower 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 3050 of file `sf_gamma.tcc`.

References `__falling_factorial()`, `__gnu_cxx::__fp_is_integer()`, and `__log_gamma()`.

9.3.2.226 `__log_gamma()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__log_gamma (
    _Tp __a )
```

Return $\log(|\Gamma(a)|)$. This will return values even for $a < 0$. To recover the sign of $\Gamma(a)$ for any argument use `__log_gamma_sign`.

Parameters

<code>__a</code>	The argument of the log of the gamma function.
------------------	--

Returns

The logarithm of the gamma function.

Definition at line 2325 of file `sf_gamma.tcc`.

References `__sin_pi()`, and `__spouge_log_gamma1p()`.

Referenced by `__beta_inc()`, `__beta_lgamma()`, `__cyl_bessel_ij_series()`, `__falling_factorial()`, `__gamma()`, `__gamma_cont_frac()`, `__gamma_series()`, `__hydrogen()`, `__hyperg()`, `__hyperg_reflect()`, `__laguerre_large_n()`, `__log_binomial()`, `__log_double_factorial()`, `__log_factorial()`, `__log_falling_factorial()`, `__log_gamma()`, `__log_rising_factorial()`, `__polygamma()`, `__polylog_exp_neg()`, `__polylog_exp_pos()`, `__riemann_zeta()`, `__rising_factorial()`, and `__sph_legendre()`.

9.3.2.227 `__log_gamma()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__log_gamma (
    std::complex<_Tp > __a )
```

Return $\log(\Gamma(a))$ for complex argument.

Parameters

<code>__a</code>	The complex argument of the log of the gamma function.
------------------	--

Returns

The complex logarithm of the gamma function.

Definition at line 2360 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`.

9.3.2.228 `__log_gamma_bernoulli()`

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (
    _Tp __x )
```

Return $\log(\Gamma(x))$ by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.

Parameters

<code>__x</code>	The argument of the log of the gamma function.
------------------	--

Returns

The logarithm of the gamma function.

Definition at line 1759 of file sf_gamma.tcc.

9.3.2.229 `__log_gamma_sign()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__log_gamma_sign (
    _Tp __a )
```

Return the sign of $\Gamma(x)$. At nonpositive integers zero is returned indicating $\Gamma(x)$ is undefined.

Parameters

<code>↵ __a</code>	The argument of the gamma function.
------------------------	-------------------------------------

Returns

The sign of the gamma function.

Definition at line 2401 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()`.

9.3.2.230 `__log_gamma_sign()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__log_gamma_sign (
    std::complex< _Tp > __a )
```

Definition at line 2413 of file sf_gamma.tcc.

9.3.2.231 __log_rising_factorial()

```
template<typename _Tp >
_Tp std::__detail::__log_rising_factorial (
    _Tp __a,
    _Tp __nu )
```

Return the logarithm of the rising factorial function or the (upper) Pochhammer symbol. The Pochhammer symbol is defined for integer order by

$$a^{\bar{\nu}} = \Gamma(a + \nu) / \Gamma(a) = \prod_{k=0}^{\nu-1} (a + k), (a)_0 = 1$$

Thus this function returns

$$\ln[a^{\bar{\nu}}] = \ln[\Gamma(a + \nu)] - \ln[\Gamma(a)], \ln[(a)_0] = 0$$

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 3199 of file sf_gamma.tcc.

References __log_gamma(), and __rising_factorial().

9.3.2.232 __log_stirling_1()

```
template<typename _Tp >
_Tp std::__detail::__log_stirling_1 (
    unsigned int __n,
    unsigned int __m )
```

Return the logarithm of the absolute value of Stirling number of the first kind.

Definition at line 318 of file sf_stirling.tcc.

9.3.2.233 __log_stirling_1_sign()

```
template<typename _Tp >
_Tp std::__detail::__log_stirling_1_sign (
    unsigned int __n,
    unsigned int __m ) [inline]
```

Return the sign of the exponent of the logarithm of the Stirling number of the first kind.

Definition at line 336 of file sf_stirling.tcc.

9.3.2.234 `__log_stirling_2()`

```
template<typename _Tp >
_Tp std::__detail::__log_stirling_2 (
    unsigned int __n,
    unsigned int __m )
```

Return the Stirling number of the second kind.

Todo Look into asymptotic solutions.

Definition at line 178 of file `sf_stirling.tcc`.

9.3.2.235 `__logint()`

```
template<typename _Tp >
_Tp std::__detail::__logint (
    const _Tp __x )
```

Return the logarithmic integral $li(x)$.

The logarithmic integral is given by

$$li(x) = Ei(\log(x))$$

Parameters

<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()`.

9.3.2.236 `__logistic_p()`

```
template<typename _Tp >
_Tp std::__detail::__logistic_p (
```

```

_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.

9.3.2.237 __logistic_pdf()

```

template<typename _Tp >
_Tp std::__detail::__logistic_pdf (
    _Tp __a,
    _Tp __b,
    _Tp __x )

```

Return the logistic probability density function.

The formula for the logistic probability density function is

$$p(x|a, b) = \frac{e^{(x-a)/b}}{b[1 + e^{(x-a)/b}]^2}$$

where $b > 0$.

Definition at line 670 of file sf_distributions.tcc.

9.3.2.238 __lognormal_p()

```

template<typename _Tp >
_Tp std::__detail::__lognormal_p (
    _Tp __mu,
    _Tp __sigma,
    _Tp __x )

```

Return the lognormal cumulative probability density function.

The formula for the lognormal cumulative probability density function is

$$F(x|\mu, \sigma) = \frac{1}{2} \left[1 - \operatorname{erf}\left(\frac{\ln x - \mu}{\sqrt{2}\sigma}\right) \right]$$

Definition at line 287 of file sf_distributions.tcc.

9.3.2.239 __lognormal_pdf()

```
template<typename _Tp >
_Tp std::__detail::__lognormal_pdf (
    _Tp __nu,
    _Tp __sigma,
    _Tp __x )
```

Return the lognormal probability density function.

The formula for the lognormal probability density function is

$$f(x|\mu, \sigma) = \frac{e^{(\ln x - \mu)^2 / 2\sigma^2}}{\sigma\sqrt{2\pi}}$$

Definition at line 259 of file sf_distributions.tcc.

9.3.2.240 __normal_p()

```
template<typename _Tp >
_Tp std::__detail::__normal_p (
    _Tp __mu,
    _Tp __sigma,
    _Tp __x )
```

Return the normal cumulative probability density function.

The formula for the normal cumulative probability density function is

$$F(x|\mu, \sigma) = \frac{1}{2} \left[1 - \operatorname{erf}\left(\frac{x - \mu}{\sqrt{2}\sigma}\right) \right]$$

Definition at line 238 of file sf_distributions.tcc.

9.3.2.241 __normal_pdf()

```
template<typename _Tp >
_Tp std::__detail::__normal_pdf (
    _Tp __mu,
    _Tp __sigma,
    _Tp __x )
```

Return the normal probability density function.

The formula for the normal probability density function is

$$f(x|\mu, \sigma) = \frac{e^{-(x-\mu)^2 / 2\sigma^2}}{\sigma\sqrt{2\pi}}$$

Definition at line 210 of file sf_distributions.tcc.

9.3.2.242 __owens_t()

```
template<typename _Tp >
_Tp std::__detail::__owens_t (
    _Tp __h,
    _Tp __a )
```

Return the Owens T function:

$$T(h, a) = \frac{1}{2\pi} \int_0^a \frac{\exp[-\frac{1}{2}h^2(1+x^2)]}{1+x^2} dx$$

This implementation is a translation of the Fortran implementation in

See also

Patefield, M. and Tandy, D. "Fast and accurate Calculation of Owen's T-Function", Journal of Statistical Software, 5 (5), 1 - 25 (2000)

Parameters

in	\leftarrow _h	The scale parameter.
in	\leftarrow _a	The integration limit.

Returns

The owens T function.

Definition at line 92 of file sf_owens_t.tcc.

References __znorm1(), and __znorm2().

9.3.2.243 __polar_pi() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polar_pi (
    _Tp __rho,
    _Tp __phi_pi ) [inline]
```

Reperiodized complex constructor.

Definition at line 401 of file sf_trig.tcc.

References __gnu_cxx::__sincos_t<_Tp>::__cos_v, __gnu_cxx::__sincos_t<_Tp>::__sin_v, and __sincos_pi().

Referenced by __cyl_bessel_jn_neg_arg(), __cyl_hankel_1(), __cyl_hankel_2(), __jacobi_theta_1(), __jacobi_theta_2(), __polylog_exp_neg(), and __polylog_exp_pos().

9.3.2.244 `__polar_pi()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polar_pi (
    _Tp __rho,
    const std::complex< _Tp > & __phi_pi ) [inline]
```

Reperiodized complex constructor.

Definition at line 413 of file `sf_trig.tcc`.

References `__gnu_cxx::__sincos_t< _Tp >::__cos_v`, `__gnu_cxx::__sincos_t< _Tp >::__sin_v`, and `__sincos_pi()`.

9.3.2.245 `__polygamma()`

```
template<typename _Tp >
_Tp std::__detail::__polygamma (
    unsigned int __m,
    _Tp __x )
```

Return the polygamma function $\psi^{(m)}(x)$.

The polygamma function is related to the Hurwitz zeta function:

$$\psi^{(m)}(x) = (-1)^{m+1} m! \zeta(m+1, x)$$

Definition at line 3465 of file `sf_gamma.tcc`.

References `__digamma()`, `__gnu_cxx::__fp_is_integer()`, `__hurwitz_zeta()`, `__log_gamma()`, and `std::__detail::__Factorial_table< _Tp >::__n`.

9.3.2.246 `__polylog()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__polylog (
    _Tp __s,
    _Tp __x )
```

Return the polylog $Li_s(x)$ for two real arguments.

The polylog is defined by

$$Li_s(x) = \sum_{k=1}^{\infty} \frac{x^k}{k^s}$$

Parameters

$_s$	The real order.
$_x$	The real argument.

Returns

The complex value of the polylogarithm.

Definition at line 1029 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()`.

9.3.2.247 __polylog() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog (
    _Tp __s,
    std::complex< _Tp > __w )
```

Return the polylog in those cases where we can calculate it.

Parameters

$_s$	The real order.
$_w$	The complex argument.

Returns

The complex value of the polylogarithm.

Definition at line 1070 of file sf_polylog.tcc.

References `__polylog()`, and `__polylog_exp()`.

9.3.2.248 __polylog_exp()

```
template<typename _Tp , typename _ArgType >
__gnu_cxx::__fp_promote_t<std::complex<_Tp>, _ArgType> std::__detail::__polylog_exp (
    _Tp __s,
    _ArgType __w )
```

This is the frontend function which calculates $Li_s(e^w)$. First we branch into different parts depending on the properties of s . This function is the same irrespective of a real or complex w , hence the template parameter `ArgType`.

Note

: I *really* wish we could return a `variant<Tp, std::complex<Tp>>`.

Parameters

\leftarrow __s	The real order.
\leftarrow __w	The real or complex argument.

Returns

The real or complex value of $Li_s(e^w)$.

Definition at line 988 of file `sf_polylog.tcc`.

References `__gnu_cxx::__fp_is_integer()`, `__polylog_exp_neg_int()`, `__polylog_exp_neg_real()`, `__polylog_exp_pos_ \leftarrow int()`, `__polylog_exp_pos_real()`, and `__polylog_exp_sum()`.

Referenced by `__bose_einstein()`, `__clausen()`, `__clausen_cl()`, `__clausen_sl()`, `__fermi_dirac()`, `__hurwitz_zeta_ \leftarrow polylog()`, and `__polylog()`.

9.3.2.249 __polylog_exp_asymp()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_asymp (
    _Tp __s,
    std::complex<_Tp > __w )
```

This function implements the asymptotic series for the polylog. It is given by

$$2 \sum_{k=0}^{\infty} \zeta(2k) w^{s-2k} / \Gamma(s-2k+1) - i\pi w^{s-1} / \Gamma(s)$$

for $Re(w) \gg 1$

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

$_s$	the real index s.
$_w$	the large complex argument w.

Returns

the value of the polylogarithm.

Definition at line 601 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()`.

9.3.2.250 `__polylog_exp_neg()` [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg (
    _Tp __s,
    std::complex< _Tp > __w )
```

This function treats the cases of negative real index s. Theoretical convergence is present for $|w| < 2\pi$. We use an optimized version of

$$Li_s(e^w) = \Gamma(1-s)(-w)^{s-1} + \frac{(2\pi)^{-s}}{\pi} A_p(w)$$

$$A_p(w) = \sum_k \frac{\Gamma(1+k-s)}{k!} \sin\left(\frac{\pi}{2}(s-k)\right) \left(\frac{w}{2\pi}\right)^k \zeta(1+k-s)$$

Parameters

$_s$	The negative real index
$_w$	The complex argument

Returns

The value of the polylogarithm.

Definition at line 365 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()`.

9.3.2.251 `__polylog_exp_neg()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg (
    int __n,
    std::complex< _Tp > __w )
```

Compute the polylogarithm for negative integer order.

$$Li_{-p}(e^w) = p!(-w)^{-(p+1)} - \sum_{k=0}^{\infty} \frac{B_{p+2k+q+1}}{(p+2k+q+1)!} \frac{(p+2k+q)!}{(2k+q)!} w^{2k+q}$$

where $q = (p+1) \bmod 2$.

Parameters

<code>__n</code>	the negative integer index $n = -p$.
<code>__w</code>	the argument w .

Returns

the value of the polylogarithm.

Definition at line 451 of file `sf_polylog.tcc`.

References `__gnu_cxx::__fp_is_equal()`, `__gnu_cxx::__fp_is_zero()`, `_Num_Euler_Maclaurin_zeta`, and `_S_Euler_Maclaurin_zeta`.

9.3.2.252 `__polylog_exp_neg_int()` [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg_int (
    int __s,
    std::complex< _Tp > __w )
```

This treats the case where s is a negative integer.

Parameters

$_s$	a negative integer.
$_w$	an arbitrary complex number

Returns

the value of the polylogarithm.

Definition at line 783 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()`.

9.3.2.253 __polylog_exp_neg_int() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg_int (
    int __s,
    _Tp __w )
```

This treats the case where `s` is a negative integer and `w` is a real.

Parameters

$_s$	a negative integer.
$_w$	the argument.

Returns

the value of the polylogarithm.

Definition at line 827 of file sf_polylog.tcc.

References `__gnu_cxx::__fp_is_zero()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_sum()`.

9.3.2.254 `__polylog_exp_neg_real()` [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg_real (
    _Tp __s,
    std::complex< _Tp > __w )
```

Return the polylog where s is a negative real value and for complex argument. Now we branch depending on the properties of w in the specific functions

Parameters

<code>__s</code>	A negative real value that does not reduce to a negative integer.
<code>__w</code>	The complex argument.

Returns

The value of the polylogarithm.

Definition at line 928 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()`.

9.3.2.255 `__polylog_exp_neg_real()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_neg_real (
    _Tp __s,
    _Tp __w )
```

Return the polylog where s is a negative real value and for real argument. Now we branch depending on the properties of w in the specific functions.

Parameters

<code>__s</code>	A negative real value.
<code>__w</code>	A real argument.

Returns

The value of the polylogarithm.

Definition at line 959 of file sf_polylog.tcc.

References `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_sum()`.

9.3.2.256 __polylog_exp_pos() [1/3]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos (
    unsigned int __s,
    std::complex< _Tp > __w )
```

This function treats the cases of positive integer index s for complex argument w .

$$Li_s(e^w) = \sum_{k=0, k! = s-1} \zeta(s-k) \frac{w^k}{k!} + [H_{s-1} - \log(-w)] \frac{w^{s-1}}{(s-1)!}$$

The radius of convergence is $|w| < 2\pi$. Note that this series involves a $\log(-x)$. gcc and Mathematica differ in their implementation of $\log(e^{i\pi})$: gcc: $\log(e^{+i\pi}) = + - i\pi$ whereas Mathematica doesn't preserve the sign in this case: $\log(e^{+i\pi}) = +i\pi$

Parameters

\leftrightarrow __s	the positive integer index.
\leftrightarrow __w	the argument.

Returns

the value of the polylogarithm.

Definition at line 217 of file sf_polylog.tcc.

References `__riemann_zeta()`.

Referenced by `__polylog_exp_pos_int()`, and `__polylog_exp_pos_real()`.

9.3.2.257 `__polylog_exp_pos()` [2/3]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos (
    unsigned int __s,
    _Tp __w )
```

This function treats the cases of positive integer index s for real argument w .

This specialization is worthwhile to catch the differing behaviour of $\log(x)$.

$$Li_s(e^w) = \sum_{k=0, k! \neq s-1} \zeta(s-k) \frac{w^k}{k!} + [H_{s-1} - \log(-w)] \frac{w^{s-1}}{(s-1)!}$$

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

<code>__s</code>	the positive integer index.
<code>__w</code>	the argument.

Returns

the value of the polylogarithm.

Definition at line 293 of file `sf_polylog.tcc`.

References `__riemann_zeta()`.

9.3.2.258 `__polylog_exp_pos()` [3/3]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos (
    _Tp __s,
    std::complex<_Tp> __w )
```

This function treats the cases of positive real index s .

The defining series is

$$Li_s(e^w) = A_s(w) + B_s(w) + \Gamma(1-s)(-w)^{s-1}$$

with

$$A_s(w) = \sum_{k=0}^m \zeta(s-k) w^k / k!$$

$$B_s(w) = \sum_{k=m+1}^{\infty} \sin(\pi/2(s-k)) \Gamma(1-s+k) \zeta(1-s+k) (w/2/\pi)^k / k!$$

Parameters

$_s$	the positive real index s.
$_w$	The complex argument w.

Returns

the value of the polylogarithm.

Definition at line 514 of file sf_polylog.tcc.

References `__gamma()`, `__log_gamma()`, `__polar_pi()`, and `__riemann_zeta()`.

9.3.2.259 `__polylog_exp_pos_int()` [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos_int (
    unsigned int __s,
    std::complex< _Tp > __w )
```

Here s is a positive integer and the function descends into the different kernels depending on w.

Parameters

$_s$	a positive integer.
$_w$	an arbitrary complex number.

Returns

The value of the polylogarithm.

Definition at line 676 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()`.

9.3.2.260 `__polylog_exp_pos_int()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos_int (
    unsigned int __s,
    _Tp __w )
```

Here *s* is a positive integer and the function descends into the different kernels depending on *w*.

Parameters

<code>__s</code>	a positive integer
<code>__w</code>	an arbitrary real argument <i>w</i>

Returns

the value of the polylogarithm.

Definition at line 735 of file `sf_polylog.tcc`.

References `__gnu_cxx::__fp_is_zero()`, `__polylog_exp_asymp()`, `__polylog_exp_pos()`, and `__polylog_exp_sum()`.

9.3.2.261 `__polylog_exp_pos_real()` [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos_real (
    _Tp __s,
    std::complex< _Tp > __w )
```

Return the polylog where *s* is a positive real value and for complex argument.

Parameters

<code>__s</code>	A positive real number.
<code>__w</code>	the complex argument.

Returns

The value of the polylogarithm.

Definition at line 854 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()`.

9.3.2.262 `__polylog_exp_pos_real()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__polylog_exp_pos_real (
    _Tp __s,
    _Tp __w )
```

Return the polylog where s is a positive real value and the argument is real.

Parameters

<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 894 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()`.

9.3.2.263 `__polylog_exp_sum()`

```
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 645 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()`.

9.3.2.264 `__prob_hermite_recur()`

```
template<typename _Tp >
__gnu_cxx::__hermite_he_t<_Tp> std::__detail::__prob_hermite_recur (
    unsigned int __n,
    _Tp __x )
```

This routine returns the Probabilists Hermite polynomial of order n : $He_n(x)$ by recursion on n .

The Probabilists 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.

The Probabilists Hermite polynomial has first and second derivatives:

$$He'_n(x) = n He_{n-1}(x)$$

and

$$He''_n(x) = n(n-1) He_{n-2}(x)$$

The Probabilists Hermite polynomial are monic and are orthogonal with respect to the weight function

$$w(x) = e^{x^2/2}$$

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 260 of file sf_hermite.tcc.

9.3.2.265 __radial_jacobi()

```
template<typename _Tp >
_Tp std::__detail::__radial_jacobi (
    unsigned int __n,
    unsigned int __m,
    _Tp __rho )
```

Return the radial polynomial $R_n^m(\rho)$ for non-negative n and m , and real radial argument ρ is a polynomial of degree $m + 2n$ in 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 are related to the Jacobi polynomials:

$$R_n^m(\rho) = (-1)^n x^m P_n^{(m,0)}(1 - 2\rho^2)$$

for $0 \leq \rho \leq 1$

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.

Zernike Polynomials: Evaluation, Quadrature, and Interpolation Philip Greengard, Kirill Serkh, Technical Report YALEU/DCS/TR-1539, February 20, 2018

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 292 of file `sf_jacobi.tcc`.

References `__jacobi_recur()`.

Referenced by `__zernike()`, `__gnu_cxx::radpolyf()`, and `__gnu_cxx::radpolyl()`.

9.3.2.266 `__radial_jacobi_zeros()`

```
template<typename _Tp >
std::vector<__gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__radial_jacobi_zeros (
    unsigned int __n,
    unsigned int __m )
```

Return a vector containing the zeros of the radial Jacobi polynomial $P_n^{(\alpha, \beta)}(1 - 2\rho^2)$.

Template Parameters

<code>_Tp</code>	The real type of the radial coordinate
------------------	--

Parameters

in	<i>n</i>	The order of the Jacobi polynomial
in	<i>alpha1</i>	The first parameter of the Jacobi polynomial
in	<i>beta1</i>	The second parameter of the Jacobi polynomial

Definition at line 324 of file `sf_jacobi.tcc`.

References `__jacobi_zeros()`.

9.3.2.267 __rice_pdf()

```
template<typename _Tp >
_Tp std::__detail::__rice_pdf (
    _Tp __nu,
    _Tp __sigma,
    _Tp __x )
```

Return the Rice probability density function.

The formula for the Rice probability density function is

$$p(x|\nu, \sigma) = \frac{x}{\sigma^2} \exp\left(-\frac{x^2 + \nu^2}{2\sigma^2}\right) I_0\left(\frac{x\nu}{\sigma^2}\right)$$

where $I_0(x)$ is the modified Bessel function of the first kind of order 0 and $\nu \geq 0$ and $\sigma > 0$.

Definition at line 186 of file sf_distributions.tcc.

References `__cyl_bessel_i()`.

9.3.2.268 __riemann_zeta()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta (
    _Tp __s )
```

Return the Riemann zeta function $\zeta(s)$.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} k^{-s} \text{ for } \Re(s) > 1 \quad \frac{(2\pi)^s}{\pi} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s) \zeta(1-s) \text{ for } \Re(s) < 1$$

Parameters

<code>__s</code>	The argument
------------------	--------------

Todo Global double sum or MacLaurin series in `riemann_zeta`?

Definition at line 761 of file sf_zeta.tcc.

References `__exp2()`, `__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()`.

9.3.2.269 __riemann_zeta_euler_maclaurin()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta_euler_maclaurin (
    _Tp __s )
```

Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.

This is a specialization of the code for the Hurwitz zeta function.

Definition at line 389 of file sf_zeta.tcc.

References `_S_Euler_Maclaurin_zeta`.

9.3.2.270 __riemann_zeta_glob()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta_glob (
    _Tp __s )
```

Definition at line 499 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()`.

9.3.2.271 __riemann_zeta_laurent()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta_laurent (
    _Tp __s )
```

Compute the Riemann zeta function $\zeta(s)$ by Laurent expansion about $s = 1$.

The Laurent expansion of the Riemann zeta function is given by:

$$\zeta(s) = \frac{1}{s-1} + \sum_{k=0}^{\infty} \frac{(-1)^k}{k!} \gamma_k (s-1)^k$$

Where γ_k are the Stieltjes constants, $\gamma_0 = \gamma_E$ the Euler-Mascheroni constant.

The Stieltjes constants can be found from a limiting process:

$$\gamma_k = \lim_{n \rightarrow \infty} \left\{ \sum_{i=1}^n \frac{(\ln i)^k}{i} - \frac{(\ln n)^{k+1}}{k+1} \right\}$$

Definition at line 312 of file sf_zeta.tcc.

References `_Num_Stieltjes`, and `_S_Stieltjes`.

Referenced by `__riemann_zeta_m_1()`.

9.3.2.272 __riemann_zeta_m_1()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta_m_1 (
    _Tp __s )
```

Return the Riemann zeta function $\zeta(s) - 1$.

Parameters

\leftarrow _s	The argument $s \neq 1$
--------------------	-------------------------

Definition at line 717 of file sf_zeta.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma(), __riemann_zeta_laurent(), __riemann_zeta_m_1_glob(), \leftarrow sin_pi(), _S_num_zetam1, and _S_zetam1.

Referenced by __polylog_exp_neg(), and __riemann_zeta().

9.3.2.273 __riemann_zeta_m_1_glob()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta_m_1_glob (
    _Tp __s )
```

Evaluate the Riemann zeta function by series for all $s \neq 1$. 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 448 of file sf_zeta.tcc.

Referenced by __riemann_zeta_glob(), and __riemann_zeta_m_1().

9.3.2.274 __riemann_zeta_product()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta_product (
    _Tp __s )
```

Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.

$$\zeta(s) = \prod_{i=1}^{\infty} \frac{1}{1 - p_i^{-s}}$$

where p_i are the prime numbers.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } \operatorname{Re} s > 1$$

For $(s) < 1$ use the reflection formula:

$$\zeta(s) = (2\pi)^s \Gamma(1-s) \zeta(1-s) / \pi$$

Parameters

$_s$	The argument
-------	--------------

Definition at line 551 of file sf_zeta.tcc.

Referenced by __riemann_zeta().

9.3.2.275 __riemann_zeta_sum()

```
template<typename _Tp >
_Tp std::__detail::__riemann_zeta_sum (
    _Tp __s )
```

Compute the Riemann zeta function $\zeta(s)$ by summation for $s > 1$.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For $s < 1$ use the reflection formula:

$$\zeta(s) = (2\pi)^s \Gamma(1-s) \zeta(1-s) / \pi$$

Definition at line 346 of file sf_zeta.tcc.

References __gamma(), and __sin_pi().

Referenced by __riemann_zeta().

9.3.2.276 __rising_factorial() [1/2]

```
template<typename _Tp >
_Tp std::__detail::__rising_factorial (
    _Tp __a,
    int __n )
```

Return the (upper) Pochhammer function or the rising factorial function. The 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 3100 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()`.

9.3.2.277 __rising_factorial() [2/2]

```
template<typename _Tp >
_Tp std::__detail::__rising_factorial (
    _Tp __a,
    _Tp __nu )
```

Return the rising factorial function or the (upper) Pochhammer function. The rising factorial function is defined by

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(\nu)$$

Many notations exist for this function:

$$(a)_{\nu}$$

, (especially in the literature of special functions),

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

Definition at line 3155 of file sf_gamma.tcc.

References `__log_gamma()`, `__log_gamma_sign()`, `std::__detail::_Factorial_table<_Tp>::__n`, and `__rising_factorial()`.

9.3.2.278 `__sin_pi()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__sin_pi (
    _Tp __x )
```

Return the reperiodized sine of argument x:

$$\sin_{\pi}(x) = \sin(\pi x)$$

Definition at line 52 of file sf_trig.tcc.

Referenced by `__cos_pi()`, `__cosh_pi()`, `__cyl_bessel_ik()`, `__cyl_bessel_jn()`, `__dirichlet_eta()`, `__gamma_reciprocal()`, `__hankel_debye()`, `__laguerre_large_n()`, `__lanczos_log_gamma1p()`, `__log_gamma()`, `__riemann_zeta()`, `__riemann_zeta_glob()`, `__riemann_zeta_m_1()`, `__riemann_zeta_sum()`, `__sin_pi()`, `__sinc_pi()`, `__sinh_pi()`, and `__spouge_log_gamma1p()`.

9.3.2.279 `__sin_pi()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sin_pi (
    std::complex< _Tp > __z )
```

Return the reperiodized sine of complex argument z:

$$\sin_{\pi}(z) = \sin(\pi z) = \sin_{\pi}(x)\cosh_{\pi}(y) + i\cos_{\pi}(x)\sinh_{\pi}(y)$$

Definition at line 187 of file sf_trig.tcc.

References `__cos_pi()`, and `__sin_pi()`.

9.3.2.280 `__sinc()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_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.

9.3.2.281 __sinc_pi()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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().

9.3.2.282 __sincos() [1/4]

```
template<typename _Tp >
__gnu_cxx::__sincos_t<_Tp> std::__detail::__sincos (
    _Tp __x ) [inline]
```

Definition at line 316 of file sf_trig.tcc.

Referenced by __sincos_pi().

9.3.2.283 __sincos() [2/4]

```
template<>
__gnu_cxx::__sincos_t<float> std::__detail::__sincos (
    float __x ) [inline]
```

Definition at line 324 of file sf_trig.tcc.

9.3.2.284 __sincos() [3/4]

```
template<>
__gnu_cxx::__sincos_t<double> std::__detail::__sincos (
    double __x ) [inline]
```

Definition at line 336 of file sf_trig.tcc.

9.3.2.285 `__sincos()` [4/4]

```
template<>
__gnu_cxx::__sincos_t<long double> std::__detail::__sincos (
    long double __x ) [inline]
```

Definition at line 348 of file `sf_trig.tcc`.

9.3.2.286 `__sincos_pi()`

```
template<typename _Tp >
__gnu_cxx::__sincos_t<_Tp> std::__detail::__sincos_pi (
    _Tp __x )
```

Reperiodized sincos.

Definition at line 360 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()`.

9.3.2.287 `__sincosint()`

```
template<typename _Tp >
std::pair<_Tp, _Tp> std::__detail::__sincosint (
    _Tp __x )
```

This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a `pair`.

The sine integral is defined by:

$$Si(x) = \int_0^x dt \frac{\sin(t)}{t}$$

The cosine integral is defined by:

$$Ci(x) = \gamma_E + \log(x) + \int_0^x dt \frac{\cos(t) - 1}{t}$$

Definition at line 226 of file `sf_trigint.tcc`.

References `__sincosint_asymp()`, `__sincosint_cont_frac()`, and `__sincosint_series()`.

9.3.2.288 __sincosint_asyp()

```
template<typename _Tp >
void std::__detail::__sincosint_asyp (
    _Tp __t,
    _Tp & __Si,
    _Tp & __Ci )
```

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.

The asymptotic series is very good for $x > 50$.

Definition at line 159 of file sf_trigint.tcc.

Referenced by __sincosint().

9.3.2.289 __sincosint_cont_frac()

```
template<typename _Tp >
void std::__detail::__sincosint_cont_frac (
    _Tp __t,
    _Tp & __Si,
    _Tp & __Ci )
```

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.

Definition at line 52 of file sf_trigint.tcc.

Referenced by __sincosint().

9.3.2.290 __sincosint_series()

```
template<typename _Tp >
void std::__detail::__sincosint_series (
    _Tp __t,
    _Tp & __Si,
    _Tp & __Ci )
```

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.

Definition at line 95 of file sf_trigint.tcc.

Referenced by __sincosint().

9.3.2.291 `__sinh_pi()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__sinh_pi (
    _Tp __x )
```

Return the reperiodized hyperbolic sine of argument x:

$$\sinh_{\pi}(x) = \sinh(\pi x)$$

Definition at line 84 of file sf_trig.tcc.

Referenced by `__sinhc_pi()`.

9.3.2.292 `__sinh_pi()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sinh_pi (
    std::complex< _Tp > __z )
```

Return the reperiodized hyperbolic sine of complex argument z:

$$\sinh_{\pi}(z) = \sinh(\pi z) = \sinh_{\pi}(x)\cos_{\pi}(y) + i\cosh_{\pi}(x)\sin_{\pi}(y)$$

Definition at line 209 of file sf_trig.tcc.

References `__cos_pi()`, and `__sin_pi()`.

9.3.2.293 `__sinhc()`

```
template<typename _Tp >
__gnu_cxx::fp_promote_t<_Tp> std::__detail::__sinhc (
    _Tp __x )
```

Return the hyperbolic sinus cardinal function

$$\operatorname{sinhc}(x) = \frac{\sinh(x)}{x}$$

.

Definition at line 97 of file sf_cardinal.tcc.

9.3.2.294 __sinhc_pi()

```
template<typename _Tp >
__gnu_cxx::fp_promote_t<_Tp> std::__detail::__sinhc_pi (
    _Tp __x )
```

Return the reperiodized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_{\pi}(x) = \frac{\sinh(\pi x)}{\pi x}$$

.

Definition at line 115 of file sf_cardinal.tcc.

References __sinh_pi().

9.3.2.295 __sinhint()


```
template<typename _Tp >
_Tp std::__detail::__sinhint (
    const _Tp __x )
```

Return the hyperbolic sine integral $Shi(x)$.

The hyperbolic sine integral is given by

$$Shi(x) = (Ei(x) + E_1(x))/2 = (Ei(x) - Ei(-x))/2$$

Parameters

 __x	The argument of the hyperbolic sine integral function.
--	--

Returns

The hyperbolic sine integral.

Definition at line 584 of file sf_expint.tcc.

References __expint_E1(), and __expint_Ei().

9.3.2.296 `__sph_bessel()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__sph_bessel (
    unsigned int __n,
    _Tp __x )
```

Return the spherical Bessel function $j_n(x)$ of order n and non-negative real argument x .

The spherical Bessel function is defined by:

$$j_n(x) = \left(\frac{\pi}{2x}\right)^{1/2} J_{n+1/2}(x)$$

Parameters

<code>__n</code>	The non-negative integral order
<code>__x</code>	The non-negative real argument

Returns

The output spherical Bessel function.

Definition at line 781 of file `sf_bessel.tcc`.

References `__sph_bessel_jn()`.

9.3.2.297 `__sph_bessel()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sph_bessel (
    unsigned int __n,
    std::complex<_Tp> __z )
```

Return the complex spherical Bessel function.

Parameters

in	<code>__n</code>	The order for which the spherical Bessel function is evaluated.
in	<code>__z</code>	The argument at which the spherical Bessel function is evaluated.

Returns

The complex spherical Bessel function.

Definition at line 1273 of file sf_hankel.tcc.

References `__sph_hankel()`.

9.3.2.298 __sph_bessel_ik()

```
template<typename _Tp >
__gnu_cxx::__sph_mod_bessel_t<unsigned int, _Tp, _Tp> std::__detail::__sph_bessel_ik (
    unsigned int __n,
    _Tp __x )
```

Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i'_n(x)$ and $k'_n(x)$ respectively.

Parameters

\leftarrow <code>__n</code>	The order of the modified spherical Bessel function.
\leftarrow <code>__x</code>	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 428 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ik()`.

9.3.2.299 __sph_bessel_jn()

```
template<typename _Tp >
__gnu_cxx::__sph_bessel_t<unsigned int, _Tp, _Tp> std::__detail::__sph_bessel_jn (
    unsigned int __n,
    _Tp __x )
```

Compute the spherical Bessel $j_n(x)$ and Neumann $n_n(x)$ functions and their first derivatives $j'_n(x)$ and $n'_n(x)$ respectively.

Parameters

\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 713 of file sf_bessel.tcc.

References `__cyl_bessel_jn()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

9.3.2.300 `__sph_bessel_jn_neg_arg()`

```
template<typename _Tp >
__gnu_cxx::__sph_bessel_t<unsigned int, _Tp, std::complex<_Tp> > std::__detail::__sph_bessel_↔
jn_neg_arg (
    unsigned int __n,
    _Tp __x )
```

Return the spherical Bessel functions and their derivatives of order ν and argument $x < 0$.

Definition at line 737 of file sf_bessel.tcc.

References `__cyl_bessel_jn_neg_arg()`.

Referenced by `__sph_hankel_1()`, and `__sph_hankel_2()`.

9.3.2.301 `__sph_hankel()`

```
template<typename _Tp >
__gnu_cxx::__sph_hankel_t<unsigned int, std::complex<_Tp>, std::complex<_Tp> > std::__detail::__↔
sph_hankel (
    unsigned int __n,
    std::complex< _Tp > __z )
```

Helper to compute complex spherical Hankel functions and their derivatives.

Parameters

in	\leftrightarrow _n	The order for which the spherical Hankel functions are evaluated.
in	\leftrightarrow _z	The argument at which the spherical Hankel functions are evaluated.

Returns

A struct containing the spherical Hankel functions of the first and second kinds and their derivatives.

Definition at line 1209 of file sf_hankel.tcc.

References `__hankel()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

9.3.2.302 __sph_hankel_1() [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sph_hankel_1 (
    unsigned int __n,
    _Tp __x )
```

Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.

The spherical Hankel function of the first kind is defined by:

$$h_n^{(1)}(x) = j_n(x) + in_n(x)$$

Parameters

\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 842 of file sf_bessel.tcc.

References `__sph_bessel_jn()`, and `__sph_bessel_jn_neg_arg()`.

9.3.2.303 `__sph_hankel_1()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sph_hankel_1 (
    unsigned int __n,
    std::complex< _Tp > __z )
```

Return the complex spherical Hankel function of the first kind.

Parameters

in	\leftrightarrow <code>__n</code>	The order for which the spherical Hankel function of the first kind is evaluated.
in	\leftrightarrow <code>__z</code>	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 1239 of file `sf_hankel.tcc`.

References `__sph_hankel()`.

9.3.2.304 `__sph_hankel_2()` [1/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sph_hankel_2 (
    unsigned int __n,
    _Tp __x )
```

Return the spherical Hankel function of the second kind $h_n^{(2)}(x)$.

The spherical Hankel function of the second kind is defined by:

$$h_n^{(2)}(x) = j_n(x) - in_n(x)$$

Parameters

\leftrightarrow <code>__n</code>	The non-negative integral order
\leftrightarrow <code>__x</code>	The non-negative real argument

Returns

The output spherical Neumann function.

Definition at line 877 of file sf_bessel.tcc.



References `__sph_bessel_jn()`, and `__sph_bessel_jn_neg_arg()`.

9.3.2.305 __sph_hankel_2() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sph_hankel_2 (
    unsigned int __n,
    std::complex< _Tp > __z )
```

Return the complex spherical Hankel function of the second kind.

Parameters

in	 __n	The order for which the spherical Hankel function of the second kind is evaluated.
in	 __z	The argument at which the spherical Hankel function of the second kind is evaluated.

Returns

The complex spherical Hankel function of the second kind.

Definition at line 1256 of file sf_hankel.tcc.

References `__sph_hankel()`.

9.3.2.306 __sph_harmonic()

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sph_harmonic (
    unsigned int __l,
    int __m,
    _Tp __theta,
    _Tp __phi )
```

Return the spherical harmonic function.

The spherical harmonic function of l , m , and θ , ϕ is defined by:

$$Y_l^m(\theta, \phi) = (-1)^m \left[\frac{(2l+1)}{4\pi} \frac{(l-m)!}{(l+m)!} \right] P_l^{|m|}(\cos \theta) \exp^{im\phi}$$

Note

$$Y_l^m(\theta, \phi) = 0 \text{ if } |m| > l.$$

Parameters

<code>__l</code>	The degree of the spherical harmonic function. $l \geq 0$.
<code>__m</code>	The order of the spherical harmonic function.
<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 371 of file `sf_legendre.tcc`.

References `__sph_legendre()`.

9.3.2.307 __sph_legendre()

```
template<typename _Tp >
_Tp std::__detail::__sph_legendre (
    unsigned int __l,
    unsigned int __m,
    _Tp __theta )
```

Return the spherical associated Legendre function.

The spherical associated Legendre function of l , m , and θ is defined as $Y_l^m(\theta, 0)$ where

$$Y_l^m(\theta, \phi) = (-1)^m \left[\frac{(2l+1)}{4\pi} \frac{(l-m)!}{(l+m)!} \right] P_l^m(\cos \theta) \exp^{im\phi}$$

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 .

Note

Unlike the case for `__assoc_legendre_p` the Condon-Shortley phase factor $(-1)^m$ is present here.

$$Y_l^m(\theta) = 0 \text{ if } m > l.$$

Parameters

<code>__l</code>	The degree of the spherical associated Legendre function. $l \geq 0$.
<code>__m</code>	The order of the spherical associated Legendre function.
<code>__theta</code>	The radian polar angle argument of the spherical associated Legendre function.

Definition at line 278 of file sf_legendre.tcc.

References `__legendre_p()`, and `__log_gamma()`.

Referenced by `__hydrogen()`, and `__sph_harmonic()`.

9.3.2.308 `__sph_neumann()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__sph_neumann (
    unsigned int __n,
    _Tp __x )
```

Return the spherical Neumann function $n_n(x)$ of order n and non-negative real argument x .

The spherical Neumann function is defined by:

$$n_n(x) = \left(\frac{\pi}{2x}\right)^{1/2} N_{n+1/2}(x)$$

Parameters

<code>↵ __n</code>	The order of the spherical Neumann function.
<code>↵ __x</code>	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Definition at line 814 of file sf_bessel.tcc.

References `__sph_bessel_jn()`.

9.3.2.309 `__sph_neumann()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__sph_neumann (
    unsigned int __n,
    std::complex<_Tp> __z )
```

Return the complex spherical Neumann function.

Parameters

in	\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 1290 of file sf_hankel.tcc.

References __sph_hankel().

9.3.2.310 __spouge_binet1p()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__spouge_binet1p (
    _Tp __z )
```

Return the Binet function $J(1+z)$ by the Spouge method. The Binet function is the log of the scaled Gamma function $\log(\Gamma^*(z))$ defined by

$$J(z) = \log(\Gamma^*(z)) = \log(\Gamma(z)) + z - \left(z - \frac{1}{2}\right) \log(z) - \log(2\pi)$$

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 1941 of file sf_gamma.tcc.

Referenced by __spouge_log_gamma1p().

9.3.2.311 __spouge_log_gamma1p()

```
template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__spouge_log_gamma1p (
    _Tp __z )
```

Return the logarithm of the gamma function $\log(\Gamma(1+z))$ by the Spouge algorithm:

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[\sqrt{2\pi} + \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

.

If the argument is real, the log of the absolute value of the Gamma function is returned. The sign to be applied to the exponential of this log Gamma can be recovered with a call to `__log_gamma_sign`.

For complex argument the fully complex log of the gamma function is returned.

See also

Spouge, J. L., Computation of the gamma, digamma, and trigamma functions. SIAM Journal on Numerical Analysis 31, 3 (1994), pp. 931-944

Parameters

<code>__z</code>	The argument of the gamma function.
------------------	-------------------------------------

Returns

The the gamma function.

Definition at line 1985 of file `sf_gamma.tcc`.

References `__sin_pi()`, and `__spouge_binet1p()`.

Referenced by `__log_gamma()`.

9.3.2.312 __stirling_1()

```
template<typename _Tp >
_Tp std::__detail::__stirling_1 (
    unsigned int __n,
    unsigned int __m )
```

Return the Stirling number of the first kind.

The Stirling numbers of the first kind are the coefficients of the Pochhammer polynomials:

$$(x)_n = \sum_{k=0}^n S_n^{(k)} x^k$$

The recursion is

$$S_{n+1}^{(m)} = S_n^{(m-1)} - nS_n^{(m)} \text{ or}$$

with starting values

$$S_0^{(0 \rightarrow m)} = 1, 0, 0, \dots, 0$$

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.

9.3.2.313 __stirling_1_recur()

```
template<typename _Tp >
_Tp std::__detail::__stirling_1_recur (
    unsigned int __n,
    unsigned int __m )
```

Return the Stirling number of the first kind by recursion. The recursion is

$$S_{n+1}^{(m)} = S_n^{(m-1)} - nS_n^{(m)} \text{ or}$$

with starting values

$$S_0^{(0 \rightarrow m)} = 1, 0, 0, \dots, 0$$

and

$$S_{0 \rightarrow n}^{(0)} = 1, 0, 0, \dots, 0$$

Definition at line 251 of file sf_stirling.tcc.

9.3.2.314 __stirling_1_series()

```
template<typename _Tp >
_Tp std::__detail::__stirling_1_series (
    unsigned int __n,
    unsigned int __m )
```

Return the Stirling number of the first kind by series expansion. N.B. This seems to be a total disaster.

Definition at line 196 of file sf_stirling.tcc.

References `__gnu_cxx::__parity()`.

9.3.2.315 __stirling_2()

```
template<typename _Tp >
_Tp std::__detail::__stirling_2 (
    unsigned int __n,
    unsigned int __m )
```

Return the Stirling number of the second kind from lookup or by series expansion.

The series is:

$$\sigma_n^{(m)} = \sum_{k=0}^m \frac{(-1)^{m-k} k^n}{(m-k)! k!}$$

Todo Find asymptotic solutions for Stirling numbers of the second kind.

Develop an iterator model for Stirling numbers of the second kind.

Definition at line 159 of file sf_stirling.tcc.

9.3.2.316 __stirling_2_recur()

```
template<typename _Tp >
_Tp std::__detail::__stirling_2_recur (
    unsigned int __n,
    unsigned int __m )
```

Return the Stirling number of the second kind by recursion. The recursion is

$$\left\{ \begin{matrix} n \\ m \end{matrix} \right\} = m \left\{ \begin{matrix} n-1 \\ m \end{matrix} \right\} + \left\{ \begin{matrix} n-1 \\ m-1 \end{matrix} \right\}$$

with starting values

$$\left\{ \begin{matrix} 0 \\ 0 \rightarrow m \end{matrix} \right\} = 1, 0, 0, \dots, 0$$

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.

9.3.2.317 __stirling_2_series()

```
template<typename _Tp >
_Tp std::__detail::__stirling_2_series (
    unsigned int __n,
    unsigned int __m )
```

Return the Stirling number of the second kind from lookup or by series expansion.

The series is:

$$\sigma_n^{(m)} = \left\{ \begin{matrix} n \\ m \end{matrix} \right\} = \sum_{k=0}^m \frac{(-1)^{m-k} k^n}{(m-k)! k!}$$

The Stirling number of the second kind is denoted by other symbols in the literature: $\sigma_n^{(m)}$, $S_n^{(m)}$ and others.

Todo Find a way to predict the maximum Stirling number for a type.

Definition at line 67 of file sf_stirling.tcc.

9.3.2.318 __student_t_p()

```
template<typename _Tp >
_Tp std::__detail::__student_t_p (
    _Tp __t,
    unsigned int __nu )
```

Return the Students T probability function.

The students T propability function is related to the incomplete beta function:

$$A(t|\nu) = 1 - I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right) A(t|\nu) =$$

Parameters

<code>__t</code>	
<code>__nu</code>	

Definition at line 444 of file sf_distributions.tcc.

References `__beta_inc()`.

9.3.2.319 __student_t_pdf()

```
template<typename _Tp >
_Tp std::__detail::__student_t_pdf (
    _Tp __t,
    unsigned int __nu )
```

Return the Students T probability density.

The students T propability density is:

$$A(t|\nu) = 1 - I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right)A(t|\nu) =$$

Parameters

<code>__t</code>	
<code>__nu</code>	

Definition at line 419 of file sf_distributions.tcc.

References `__gamma()`.

9.3.2.320 __student_t_q()

```
template<typename _Tp >
_Tp std::__detail::__student_t_q (
    _Tp __t,
    unsigned int __nu )
```

Return the complement of the Students T probability function.

The complement of the students T propability function is:

$$A_c(t|\nu) = I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right) = 1 - A(t|\nu)$$

Parameters

<code>__t</code>	
<code>__nu</code>	

Definition at line 467 of file sf_distributions.tcc.

References `__beta_inc()`.

9.3.2.321 `__tan_pi()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__tan_pi (
    _Tp __x )
```

Return the reperiodized tangent of argument x:

$$\tan_p i(x) = \tan(\pi x)$$

Definition at line 153 of file sf_trig.tcc.

Referenced by `__digamma()`, `__tan_pi()`, and `__tanh_pi()`.

9.3.2.322 `__tan_pi()` [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__tan_pi (
    std::complex< _Tp > __z )
```

Return the reperiodized tangent of complex argument z:

$$\tan_\pi(z) = \tan(\pi z) = \frac{\tan_\pi(x) + i \tanh_\pi(y)}{1 - i \tan_\pi(x) \tanh_\pi(y)}$$

Definition at line 275 of file sf_trig.tcc.

References `__tan_pi()`.

9.3.2.323 `__tanh_pi()` [1/2]

```
template<typename _Tp >
_Tp std::__detail::__tanh_pi (
    _Tp __x )
```

Return the reperiodized hyperbolic tangent of argument x:

$$\tanh_\pi(x) = \tanh(\pi x)$$

Definition at line 169 of file sf_trig.tcc.

9.3.2.324 __tanh_pi() [2/2]

```
template<typename _Tp >
std::complex<_Tp> std::__detail::__tanh_pi (
    std::complex<_Tp > __z )
```

Return the reperiodized hyperbolic tangent of complex argument z:

$$\tanh_{\pi}(z) = \tanh(\pi z) = \frac{\tanh_{\pi}(x) + i \tan_{\pi}(y)}{1 + i \tanh_{\pi}(x) \tan_{\pi}(y)}$$

Definition at line 298 of file sf_trig.tcc.

References __tan_pi().

9.3.2.325 __tgamma()

```
template<typename _Tp >
_Tp std::__detail::__tgamma (
    _Tp __a,
    _Tp __x )
```

Return the upper incomplete gamma function. The lower incomplete gamma function is defined by

$$\Gamma(a, x) = \int_x^{\infty} e^{-t} t^{a-1} dt (a > 0)$$

.

Definition at line 2903 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma_cont_frac(), and __gamma_series().

Referenced by __gamma_q().

9.3.2.326 __tgamma_lower()

```
template<typename _Tp >
_Tp std::__detail::__tgamma_lower (
    _Tp __a,
    _Tp __x )
```

Return the lower incomplete gamma function. The lower incomplete gamma function is defined by

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

.

Definition at line 2868 of file sf_gamma.tcc.

References __gnu_cxx::__fp_is_integer(), __gamma_cont_frac(), and __gamma_series().

Referenced by __gamma_p().

9.3.2.327 __theta_1()

```
template<typename _Tp >
_Tp std::__detail::__theta_1 (
    _Tp __nu,
    _Tp __x )
```

Return the exponential theta-1 function of period `nu` and argument `x`.

The exponential theta-1 function is defined by

$$\theta_1(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{k=-\infty}^{+\infty} (-1)^k \exp\left(\frac{-(\nu + k - 1/2)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 212 of file `sf_theta.tcc`.

References `__gnu_cxx::__fp_is_zero()`, and `__theta_2()`.

Referenced by `__theta_s()`.

9.3.2.328 __theta_2()

```
template<typename _Tp >
_Tp std::__detail::__theta_2 (
    _Tp __nu,
    _Tp __x )
```

Return the exponential theta-2 function of period `nu` and argument `x`.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{k=-\infty}^{+\infty} (-1)^k \exp\left(\frac{-(\nu + k)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 184 of file sf_theta.tcc.

References `__theta_2_asymp()`, and `__theta_2_sum()`.

Referenced by `__theta_1()`, and `__theta_c()`.

9.3.2.329 `__theta_2_asymp()`

```
template<typename _Tp >
_Tp std::__detail::__theta_2_asymp (
    _Tp __nu,
    _Tp __x )
```

Compute and return the exponential θ_2 function by asymptotic series expansion:

$$\theta_2(\nu, x) = 2 \sum_{k=0}^{\infty} e^{-((k+1/2)\pi)^2 x} \cos((2k+1)\nu\pi)$$

Definition at line 120 of file sf_theta.tcc.

Referenced by `__theta_2()`.

9.3.2.330 `__theta_2_sum()`

```
template<typename _Tp >
_Tp std::__detail::__theta_2_sum (
    _Tp __nu,
    _Tp __x )
```

Compute and return the exponential θ_2 function by series expansion:

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{k=-\infty}^{\infty} (-1)^k e^{-(\nu+k)^2/x}$$

Definition at line 56 of file sf_theta.tcc.

Referenced by `__theta_2()`.

9.3.2.331 `__theta_3()`

```
template<typename _Tp >
_Tp std::__detail::__theta_3 (
    _Tp __nu,
    _Tp __x )
```

Return the exponential theta-3 function of period `nu` and argument `x`.

The exponential theta-3 function is defined by

$$\theta_3(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{k=-\infty}^{+\infty} \exp\left(\frac{-(\nu+k)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 240 of file sf_theta.tcc.

References `__theta_3_asymp()`, and `__theta_3_sum()`.

Referenced by `__theta_4()`, and `__theta_d()`.

9.3.2.332 `__theta_3_asymp()`

```
template<typename _Tp >
_Tp std::__detail::__theta_3_asymp (
    _Tp __nu,
    _Tp __x )
```

Compute and return the exponential θ_3 function by asymptotic series expansion:

$$\theta_3(\nu, x) = 1 + 2 \sum_{k=1}^{\infty} e^{-(k\pi)^2 x} \cos(2k\nu\pi)$$

Definition at line 150 of file sf_theta.tcc.

Referenced by `__theta_3()`.

9.3.2.333 `__theta_3_sum()`

```
template<typename _Tp >
_Tp std::__detail::__theta_3_sum (
    _Tp __nu,
    _Tp __x )
```

Compute and return the exponential θ_3 function by series expansion:

$$\theta_3(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{k=-\infty}^{\infty} e^{-(\nu+k)^2/x}$$

Definition at line 89 of file sf_theta.tcc.

Referenced by `__theta_3()`.

9.3.2.334 __theta_4()

```
template<typename _Tp >
_Tp std::__detail::__theta_4 (
    _Tp __nu,
    _Tp __x )
```

Return the exponential theta-4 function of period `nu` and argument `x`.

The exponential theta-4 function is defined by

$$\theta_4(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{k=-\infty}^{+\infty} (-1)^k \exp\left(\frac{-(\nu + k)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 268 of file `sf_theta.tcc`.

References `__theta_3()`.

Referenced by `__theta_n()`.

9.3.2.335 __theta_c()

```
template<typename _Tp >
_Tp std::__detail::__theta_c (
    _Tp __k,
    _Tp __x )
```

Return the Neville θ_c function

$$\theta_c(k, x) = \sqrt{\frac{\pi}{2kK(k)}} \theta_1\left(q(k), \frac{\pi x}{2K(k)}\right)$$

Definition at line 382 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_2()`.

9.3.2.336 __theta_d()

```
template<typename _Tp >
_Tp std::__detail::__theta_d (
    _Tp __k,
    _Tp __x )
```

Return the Neville θ_d function

$$\theta_d(k, x) = \sqrt{\frac{\pi}{2K(k)}} \theta_3 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

Definition at line 411 of file sf_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_3()`.

9.3.2.337 __theta_n()

```
template<typename _Tp >
_Tp std::__detail::__theta_n (
    _Tp __k,
    _Tp __x )
```

Return the Neville θ_n function

The Neville theta-n function is defined by

$$\theta_n(k, x) = \sqrt{\frac{\pi}{2k'K(k)}} \theta_4 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

Definition at line 442 of file sf_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_4()`.

9.3.2.338 __theta_s()

```
template<typename _Tp >
_Tp std::__detail::__theta_s (
    _Tp __k,
    _Tp __x )
```

Return the Neville θ_s function

$$\theta_s(k, x) = \sqrt{\frac{\pi}{2kk'K(k)}} \theta_1 \left(q(k), \frac{\pi x}{2K(k)} \right)$$

Definition at line 352 of file sf_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_1()`.

9.3.2.339 __tricomi_u()

```
template<typename _Tp >
_Tp std::__detail::__tricomi_u (
    _Tp __a,
    _Tp __c,
    _Tp __x )
```

Return the Tricomi confluent hypergeometric function

$$U(a, c, x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a; c; x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1; 2-c; x)$$

.

Parameters

\longleftrightarrow __a	The <i>numerator</i> parameter.
\longleftrightarrow __c	The <i>denominator</i> parameter.
\longleftrightarrow __x	The argument of the confluent hypergeometric function.

Returns

The Tricomi confluent hypergeometric function.

Definition at line 402 of file sf_hyperg.tcc.

References __tricomi_u_naive().

9.3.2.340 __tricomi_u_naive()

```
template<typename _Tp >
_Tp std::__detail::__tricomi_u_naive (
    _Tp __a,
    _Tp __c,
    _Tp __x )
```

Return the Tricomi confluent hypergeometric function

$$U(a, c, x) = \frac{\Gamma(1-c)}{\Gamma(a-c+1)} {}_1F_1(a; c; x) + \frac{\Gamma(c-1)}{\Gamma(a)} x^{1-c} {}_1F_1(a-c+1; 2-c; x)$$

.

Parameters

\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 368 of file sf_hyperg.tcc.

References `__conf_hyperg()`, `__gnu_cxx::__fp_is_integer()`, and `__gnu_cxx::tgamma()`.

Referenced by `__tricomi_u()`.

9.3.2.341 `__weibull_p()`

```
template<typename _Tp >
_Tp std::__detail::__weibull_p (
    _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.

9.3.2.342 `__weibull_pdf()`

```
template<typename _Tp >
_Tp std::__detail::__weibull_pdf (
    _Tp __a,
    _Tp __b,
    _Tp __x )
```

Return the Weibull probability density function.

The formula for the Weibull probability density function is

$$f(x|a, b) = \frac{a}{b} \left(\frac{x}{b}\right)^{a-1} \exp - \left(\frac{x}{b}\right)^a \text{ for } x \geq 0$$

Definition at line 374 of file sf_distributions.tcc.

9.3.2.343 __zernike()

```
template<typename _Tp >
__gnu_cxx::fp_promote_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

[__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 371 of file sf_jacobi.tcc.

References [__radial_jacobi](#)).

9.3.2.344 __znorm1()

```
template<typename _Tp >
_Tp std::__detail::__znorm1 (
    _Tp __x )
```

Definition at line 58 of file sf_owens_t.tcc.

Referenced by `__owens_t()`.

9.3.2.345 `__znorm2()`

```
template<typename _Tp >
_Tp std::__detail::__znorm2 (
    _Tp __x )
```

Definition at line 47 of file sf_owens_t.tcc.

Referenced by `__owens_t()`.

9.3.3 Variable Documentation

9.3.3.1 `__max_FGH`

```
template<typename _Tp >
constexpr int std::__detail::__max_FGH = \_Airy\_series<_Tp>::_N_FGH
```

Definition at line 178 of file sf_airy.tcc.

9.3.3.2 `__max_FGH< double >`

```
template<>
constexpr int std::\_\_detail::\_\_max\_FGH< double > = 79
```

Definition at line 184 of file sf_airy.tcc.

9.3.3.3 `__max_FGH< float >`

```
template<>
constexpr int std::\_\_detail::\_\_max\_FGH< float > = 15
```

Definition at line 181 of file sf_airy.tcc.

9.3.3.4 _Num_Euler_Maclaurin_zeta

```
constexpr size_t std::__detail::_Num_Euler_Maclaurin_zeta = 100
```

Coefficients for Euler-Maclaurin summation of zeta functions.

$$B_{2j}/(2j)!$$

where B_k are the Bernoulli numbers.

Definition at line 117 of file sf_zeta.tcc.

Referenced by __polylog_exp_neg().

9.3.3.5 _Num_Stieltjes

```
constexpr size_t std::__detail::_Num_Stieltjes = 21
```

Coefficients for the expansion of the Riemann zeta function:

$$\zeta(s) = \frac{1}{s-1} + \sum_{n=0}^{\infty} \frac{(-1)^n}{n!} \gamma_n (s-1)^n$$

$\gamma_0 = \gamma_E$ the Euler-Masceroni constant.

<http://www.plouffe.fr/simon/constants/stieltjesgamma.txt>

Definition at line 83 of file sf_zeta.tcc.

Referenced by __riemann_zeta_laurent().

9.3.3.6 _S_double_factorial_table

```
constexpr _Factorial_table<long double> std::__detail::_S_double_factorial_table[301]
```

Definition at line 280 of file sf_gamma.tcc.

Referenced by __double_factorial(), __log_double_factorial(), and __log_factorial().

9.3.3.7 _S_Euler_Maclaurin_zeta

```
constexpr long double std::__detail::_S_Euler_Maclaurin_zeta[_Num_Euler_Maclaurin_zeta]
```

Definition at line 120 of file sf_zeta.tcc.

Referenced by __hurwitz_zeta_euler_maclaurin(), __polylog_exp_neg(), and __riemann_zeta_euler_maclaurin().

9.3.3.8 `_S_factorial_table`

```
constexpr \_Factorial\_table<long double> std::__detail::_S_factorial_table[171]
```

Definition at line 90 of file `sf_gamma.tcc`.

Referenced by `__factorial()`, `__gamma()`, `__gamma_reciprocal()`, `__log_factorial()`, and `__log_gamma()`.

9.3.3.9 `_S_harmonic_denom`

```
constexpr unsigned long long std::__detail::_S_harmonic_denom[\_S\_num\_harmonic\_number]
```

Definition at line 3252 of file `sf_gamma.tcc`.

Referenced by `__harmonic_number()`.

9.3.3.10 `_S_harmonic_numer`

```
constexpr unsigned long long std::__detail::_S_harmonic_numer[\_S\_num\_harmonic\_number]
```

Definition at line 3219 of file `sf_gamma.tcc`.

Referenced by `__harmonic_number()`.

9.3.3.11 `_S_neg_double_factorial_table`

```
constexpr \_Factorial\_table<long double> std::__detail::_S_neg_double_factorial_table[999]
```

Definition at line 601 of file `sf_gamma.tcc`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

9.3.3.12 `_S_num_double_factorials`

```
template<typename _Tp >  
constexpr std::size_t std::__detail::_S_num_double_factorials = 0
```

Definition at line 265 of file `sf_gamma.tcc`.

9.3.3.13 `_S_num_double_factorials< double >`

```
template<>
constexpr std::size_t std::__detail::_S_num_double_factorials< double > = 301
```

Definition at line 270 of file sf_gamma.tcc.

9.3.3.14 `_S_num_double_factorials< float >`

```
template<>
constexpr std::size_t std::__detail::_S_num_double_factorials< float > = 57
```

Definition at line 268 of file sf_gamma.tcc.

9.3.3.15 `_S_num_double_factorials< long double >`

```
template<>
constexpr std::size_t std::__detail::_S_num_double_factorials< long double > = 301
```

Definition at line 272 of file sf_gamma.tcc.

9.3.3.16 `_S_num_factorials`

```
template<typename _Tp >
constexpr std::size_t std::__detail::_S_num_factorials = 0
```

Definition at line 75 of file sf_gamma.tcc.

9.3.3.17 `_S_num_factorials< double >`

```
template<>
constexpr std::size_t std::__detail::_S_num_factorials< double > = 171
```

Definition at line 80 of file sf_gamma.tcc.

9.3.3.18 `_S_num_factorials< float >`

```
template<>
constexpr std::size_t std::__detail::_S_num_factorials< float > = 35
```

Definition at line 78 of file `sf_gamma.tcc`.

9.3.3.19 `_S_num_factorials< long double >`

```
template<>
constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171
```

Definition at line 82 of file `sf_gamma.tcc`.

9.3.3.20 `_S_num_harmonic_numer`

```
constexpr unsigned long long std::__detail::_S_num_harmonic_numer = 29
```

Definition at line 3216 of file `sf_gamma.tcc`.

Referenced by `__harmonic_number()`.

9.3.3.21 `_S_num_neg_double_factorials`

```
template<typename _Tp >
constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0
```

Definition at line 585 of file `sf_gamma.tcc`.

9.3.3.22 `_S_num_neg_double_factorials< double >`

```
template<>
constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150
```

Definition at line 590 of file `sf_gamma.tcc`.

9.3.3.23 `_S_num_neg_double_factorials< float >`

```
template<>
constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27
```

Definition at line 588 of file `sf_gamma.tcc`.

9.3.3.24 `_S_num_neg_double_factorials< long double >`

```
template<>
constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999
```

Definition at line 592 of file `sf_gamma.tcc`.

9.3.3.25 `_S_num_zetam1`

```
constexpr size_t std::__detail::_S_num_zetam1 = 121
```

Table of $\zeta(n) - 1$ from 0 - 120. MPFR @ 128 bits precision.

Definition at line 580 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

9.3.3.26 `_S_Stieltjes`

```
constexpr long double std::__detail::_S_Stieltjes[_Num_Stieltjes]
```

Initial value:

```
{
    +0.5772156649015328606065120900824024310421593359L,
    -0.0728158454836767248605863758749013191377363383L,
    -0.0096903631928723184845303860352125293590658061L,
    +0.0020538344203033458661600465427533842857158044L,
    +0.0023253700654673000574681701775260680009044694L,
    +0.0007933238173010627017533348774444448307315394L,
    -0.0002387693454301996098724218419080042777837151L,
    -0.0005272895670577510460740975054788582819962534L,
    -0.0003521233538030395096020521650012087417291805L,
    -0.0000343947744180880481779146237982273906207895L,
    +0.0002053328149090647946837222892370653029598537L,
    +0.0002701844395439035266729020820679556738278420L,
    +0.0001672729121051401933535015433411834466078066L,
    -0.0000274638066037601588600076036933551815267853L,
    -0.0002092092620592999458371396973445849578315442L,
    -0.0002834686553202414466429344749971269770687029L,
    -0.0001996968583089697747077845632032403919157649L,
    +0.0000262770371099183366994665976305101228160786L,
    +0.0003073684081492528265927547519486256455238112L,
    +0.0005036054530473556290555964377171600353212698L,
    +0.0004663435615115594494005948244335505251131434L,
}
```

Definition at line 86 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_laurent()`.

9.3.3.27 `_S_zetam1`

```
constexpr long double std::__detail::_S_zetam1[\_S\_num\_zetam1]
```

Definition at line 584 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

Chapter 10

Class Documentation

10.1 `__gnu_cxx::__airy_t<_Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of this Airy function state.

Public Attributes

- `_Tp __Ai_deriv`
The derivative of the Airy function Ai.
- `_Tp __Ai_value`
The value of the Airy function Ai.
- `_Tp __Bi_deriv`
The derivative of the Airy function Bi.
- `_Tp __Bi_value`
The value of the Airy function Bi.
- `_Tx __x_arg`
The argument of the Airy fuctions.

10.1.1 Detailed Description

```
template<typename _Tx, typename _Tp>  
struct __gnu_cxx::__airy_t<_Tx, _Tp>
```

Definition at line 352 of file `specfun_state.h`.

10.1.2 Member Function Documentation

10.1.2.1 __Wronskian()

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__airy_t< _Tx, _Tp >::__Wronskian ( ) const [inline]
```

Return the Wronskian of this Airy function state.

Definition at line 370 of file specfun_state.h.

10.1.3 Member Data Documentation

10.1.3.1 __Ai_deriv

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__airy_t< _Tx, _Tp >::__Ai_deriv
```

The derivative of the Airy function Ai.

Definition at line 361 of file specfun_state.h.

10.1.3.2 __Ai_value

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__airy_t< _Tx, _Tp >::__Ai_value
```

The value of the Airy function Ai.

Definition at line 358 of file specfun_state.h.

10.1.3.3 __Bi_deriv

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__airy_t< _Tx, _Tp >::__Bi_deriv
```

The derivative of the Airy function Bi.

Definition at line 367 of file specfun_state.h.

10.1.3.4 __Bi_value

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__airy_t< _Tx, _Tp >::__Bi_value
```

The value of the Airy function Bi.

Definition at line 364 of file specfun_state.h.

10.1.3.5 __x_arg

```
template<typename _Tx , typename _Tp >
_Tx __gnu_cxx::__airy_t< _Tx, _Tp >::__x_arg
```

The argument of the Airy fuctions.

Definition at line 355 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.2 __gnu_cxx::__chebyshev_t_t<_Tp> Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- [_Tp deriv](#) () const
- [_Tp deriv2](#) () const

Public Attributes

- [std::size_t __n](#)
- [_Tp __T_n](#)
- [_Tp __T_nm1](#)
- [_Tp __T_nm2](#)
- [_Tp __x](#)

10.2.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__chebyshev_t_t< _Tp >
```

A struct to store the state of a Chebyshev polynomial of the first kind.

Definition at line 201 of file specfun_state.h.

10.2.2 Member Function Documentation

10.2.2.1 deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_t_t< _Tp >::deriv ( ) const [inline]
```

Definition at line 210 of file specfun_state.h.

10.2.2.2 deriv2()

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_t_t< _Tp >::deriv2 ( ) const [inline]
```

Definition at line 214 of file specfun_state.h.

10.2.3 Member Data Documentation

10.2.3.1 __n

```
template<typename _Tp >
std::size_t __gnu_cxx::__chebyshev_t_t< _Tp >::__n
```

Definition at line 203 of file specfun_state.h.

10.2.3.2 `__T_n`

```
template<typename _Tp >
__Tp __gnu_cxx::__chebyshev_t_t< _Tp >::__T_n
```

Definition at line 205 of file `specfun_state.h`.

10.2.3.3 `__T_nm1`

```
template<typename _Tp >
__Tp __gnu_cxx::__chebyshev_t_t< _Tp >::__T_nm1
```

Definition at line 206 of file `specfun_state.h`.

10.2.3.4 `__T_nm2`

```
template<typename _Tp >
__Tp __gnu_cxx::__chebyshev_t_t< _Tp >::__T_nm2
```

Definition at line 207 of file `specfun_state.h`.

10.2.3.5 `__x`

```
template<typename _Tp >
__Tp __gnu_cxx::__chebyshev_t_t< _Tp >::__x
```

Definition at line 204 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- `include/bits/specfun_state.h`

10.3 `__gnu_cxx::__chebyshev_u_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp deriv () const`

Public Attributes

- `std::size_t __n`
- `_Tp __U_n`
- `_Tp __U_nm1`
- `_Tp __U_nm2`
- `_Tp __x`

10.3.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__chebyshev_u_t< _Tp >
```

A struct to store the state of a Chebyshev polynomial of the second kind.

Definition at line 228 of file `specfun_state.h`.

10.3.2 Member Function Documentation

10.3.2.1 deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_u_t< _Tp >::deriv ( ) const [inline]
```

Definition at line 237 of file `specfun_state.h`.

10.3.3 Member Data Documentation

10.3.3.1 __n

```
template<typename _Tp >
std::size_t __gnu_cxx::__chebyshev_u_t< _Tp >::__n
```

Definition at line 230 of file `specfun_state.h`.

10.3.3.2 `__U_n`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_u_t< _Tp >::__U_n
```

Definition at line 232 of file `specfun_state.h`.

10.3.3.3 `__U_nm1`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_u_t< _Tp >::__U_nm1
```

Definition at line 233 of file `specfun_state.h`.

10.3.3.4 `__U_nm2`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_u_t< _Tp >::__U_nm2
```

Definition at line 234 of file `specfun_state.h`.

10.3.3.5 `__x`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_u_t< _Tp >::__x
```

Definition at line 231 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- `include/bits/specfun_state.h`

10.4 `__gnu_cxx::__chebyshev_v_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp deriv () const`

Public Attributes

- `std::size_t __n`
- `_Tp __V_n`
- `_Tp __V_nm1`
- `_Tp __V_nm2`
- `_Tp __x`

10.4.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__chebyshev_v_t< _Tp >
```

A struct to store the state of a Chebyshev polynomial of the third kind.

Definition at line 248 of file `specfun_state.h`.

10.4.2 Member Function Documentation

10.4.2.1 deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_v_t< _Tp >::deriv ( ) const [inline]
```

Definition at line 257 of file `specfun_state.h`.

10.4.3 Member Data Documentation

10.4.3.1 __n

```
template<typename _Tp >
std::size_t __gnu_cxx::__chebyshev_v_t< _Tp >::__n
```

Definition at line 250 of file `specfun_state.h`.

10.4.3.2 `__V_n`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_v_t< _Tp >::__V_n
```

Definition at line 252 of file `specfun_state.h`.

10.4.3.3 `__V_nm1`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_v_t< _Tp >::__V_nm1
```

Definition at line 253 of file `specfun_state.h`.

10.4.3.4 `__V_nm2`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_v_t< _Tp >::__V_nm2
```

Definition at line 254 of file `specfun_state.h`.

10.4.3.5 `__x`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_v_t< _Tp >::__x
```

Definition at line 251 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- `include/bits/specfun_state.h`

10.5 `__gnu_cxx::__chebyshev_w_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp deriv () const`

Public Attributes

- `std::size_t __n`
- `_Tp __W_n`
- `_Tp __W_nm1`
- `_Tp __W_nm2`
- `_Tp __x`

10.5.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__chebyshev_w_t<_Tp>
```

A struct to store the state of a Chebyshev polynomial of the fourth kind.

Definition at line 270 of file `specfun_state.h`.

10.5.2 Member Function Documentation

10.5.2.1 `deriv()`

```
template<typename _Tp>
_Tp __gnu_cxx::__chebyshev_w_t<_Tp>::deriv ( ) const [inline]
```

Definition at line 279 of file `specfun_state.h`.

10.5.3 Member Data Documentation

10.5.3.1 `__n`

```
template<typename _Tp>
std::size_t __gnu_cxx::__chebyshev_w_t<_Tp>::__n
```

Definition at line 272 of file `specfun_state.h`.

10.5.3.2 `__W_n`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_w_t< _Tp >::__W_n
```

Definition at line 274 of file `specfun_state.h`.

10.5.3.3 `__W_nm1`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_w_t< _Tp >::__W_nm1
```

Definition at line 275 of file `specfun_state.h`.

10.5.3.4 `__W_nm2`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_w_t< _Tp >::__W_nm2
```

Definition at line 276 of file `specfun_state.h`.

10.5.3.5 `__x`

```
template<typename _Tp >
_Tp __gnu_cxx::__chebyshev_w_t< _Tp >::__x
```

Definition at line 273 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- `include/bits/specfun_state.h`

10.6 `__gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of this cylindrical Bessel function state.

Public Attributes

- `_Tp __J_deriv`
The derivative of the Bessel function of the first kind.
- `_Tp __J_value`
The value of the Bessel function of the first kind.
- `_Tp __N_deriv`
The derivative of the Bessel function of the second kind.
- `_Tp __N_value`
The value of the Bessel function of the second kind.
- `_Tnu __nu_arg`
The real order of the cylindrical Bessel functions.
- `_Tx __x_arg`
The argument of the cylindrical Bessel functions.

10.6.1 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 405 of file `specfun_state.h`.

10.6.2 Member Function Documentation

10.6.2.1 __Wronskian()

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__Wronskian ( ) const [inline]
```

Return the Wronskian of this cylindrical Bessel function state.

Definition at line 426 of file `specfun_state.h`.

10.6.3 Member Data Documentation

10.6.3.1 `__J_deriv`

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__J_deriv
```

The derivative of the Bessel function of the first kind.

Definition at line 417 of file `specfun_state.h`.

10.6.3.2 `__J_value`

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__J_value
```

The value of the Bessel function of the first kind.

Definition at line 414 of file `specfun_state.h`.

10.6.3.3 `__N_deriv`

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__N_deriv
```

The derivative of the Bessel function of the second kind.

Definition at line 423 of file `specfun_state.h`.

10.6.3.4 `__N_value`

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__N_value
```

The value of the Bessel function of the second kind.

Definition at line 420 of file `specfun_state.h`.

10.6.3.5 __nu_arg

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tnu __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__nu_arg
```

The real order of the cylindrical Bessel functions.

Definition at line 408 of file `specfun_state.h`.

10.6.3.6 __x_arg

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tx __gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >::__x_arg
```

The argument of the cylindrical Bessel functions.

Definition at line 411 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- `include/bits/specfun_state.h`

10.7 __gnu_cxx::__cyl_coulomb_t<_Teta, _Trho, _Tp> Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of this Coulomb function state.

Public Attributes

- `_Teta __eta_arg`
The real parameter of the Coulomb functions.
- `_Tp __F_deriv`
The derivative of the regular Coulomb function.
- `_Tp __F_value`
The value of the regular Coulomb function.
- `_Tp __G_deriv`
The derivative of the irregular Coulomb function.
- `_Tp __G_value`
The value of the irregular Coulomb function.
- `unsigned int __l`
The nonnegative order of the Coulomb functions.
- `_Trho __rho_arg`
The argument of the Coulomb functions.

10.7.1 Detailed Description

```
template<typename _Teta, typename _Trho, typename _Tp>
struct __gnu_cxx::__cyl_coulomb_t<_Teta, _Trho, _Tp>
```

This struct captures the state of the Coulomb functions at a given order and argument.

Definition at line 435 of file `specfun_state.h`.

10.7.2 Member Function Documentation

10.7.2.1 `__Wronskian()`

```
template<typename _Teta , typename _Trho , typename _Tp >
_Tp __gnu_cxx::__cyl_coulomb_t<_Teta, _Trho, _Tp>::__Wronskian ( ) const [inline]
```

Return the Wronskian of this Coulomb function state.

Definition at line 459 of file `specfun_state.h`.

10.7.3 Member Data Documentation

10.7.3.1 `__eta_arg`

```
template<typename _Teta , typename _Trho , typename _Tp >
_Teta __gnu_cxx::__cyl_coulomb_t<_Teta, _Trho, _Tp>::__eta_arg
```

The real parameter of the Coulomb functions.

Definition at line 441 of file `specfun_state.h`.

10.7.3.2 `__F_deriv`

```
template<typename _Teta , typename _Trho , typename _Tp >
_Tp __gnu_cxx::__cyl_coulomb_t<_Teta, _Trho, _Tp>::__F_deriv
```

The derivative of the regular Coulomb function.

Definition at line 450 of file `specfun_state.h`.

10.7.3.3 __F_value

```
template<typename _Teta , typename _Trho , typename _Tp >  
_Tp __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__F_value
```

The value of the regular Coulomb function.

Definition at line 447 of file specfun_state.h.

10.7.3.4 __G_deriv

```
template<typename _Teta , typename _Trho , typename _Tp >  
_Tp __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__G_deriv
```

The derivative of the irregular Coulomb function.

Definition at line 456 of file specfun_state.h.

10.7.3.5 __G_value

```
template<typename _Teta , typename _Trho , typename _Tp >  
_Tp __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__G_value
```

The value of the irregular Coulomb function.

Definition at line 453 of file specfun_state.h.

10.7.3.6 __l

```
template<typename _Teta , typename _Trho , typename _Tp >  
unsigned int __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__l
```

The nonnegative order of the Coulomb functions.

Definition at line 438 of file specfun_state.h.

10.7.3.7 `__rho_arg`

```
template<typename _Teta , typename _Trho , typename _Tp >
_Trho __gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >::__rho_arg
```

The argument of the Coulomb functions.

Definition at line 444 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.8 `__gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of this cylindrical Hankel function state.

Public Attributes

- `_Tp __H1_deriv`
The derivative of the cylindrical Hankel function of the first kind.
- `_Tp __H1_value`
The value of the cylindrical Hankel function of the first kind.
- `_Tp __H2_deriv`
The derivative of the cylindrical Hankel function of the second kind.
- `_Tp __H2_value`
The value of the cylindrical Hankel function of the second kind.
- `_Tnu __nu_arg`
The real order of the cylindrical Hankel functions.
- `_Tx __x_arg`
The argument of the modified Hankel functions.

10.8.1 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 502 of file `specfun_state.h`.

10.8.2 Member Function Documentation

10.8.2.1 __Wronskian()

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__Wronskian ( ) const [inline]
```

Return the Wronskian of this cylindrical Hankel function state.

Definition at line 523 of file specfun_state.h.

10.8.3 Member Data Documentation

10.8.3.1 __H1_deriv

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__H1_deriv
```

The derivative of the cylindrical Hankel function of the first kind.

Definition at line 514 of file specfun_state.h.

10.8.3.2 __H1_value

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__H1_value
```

The value of the cylindrical Hankel function of the first kind.

Definition at line 511 of file specfun_state.h.

10.8.3.3 __H2_deriv

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__H2_deriv
```

The derivative of the cylindrical Hankel function of the second kind.

Definition at line 520 of file specfun_state.h.

10.8.3.4 `__H2_value`

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tp __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__H2_value
```

The value of the cylindrical Hankel function of the second kind.

Definition at line 517 of file `specfun_state.h`.

10.8.3.5 `__nu_arg`

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tnu __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__nu_arg
```

The real order of the cylindrical Hankel functions.

Definition at line 505 of file `specfun_state.h`.

10.8.3.6 `__x_arg`

```
template<typename _Tnu, typename _Tx, typename _Tp>
_Tx __gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >::__x_arg
```

The argument of the modified Hankel functions.

Definition at line 508 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.9 `__gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of this modified cylindrical Bessel function state.

Public Attributes

- [_Tp __I_deriv](#)
The derivative of the modified cylindrical Bessel function of the first kind.
- [_Tp __I_value](#)
The value of the modified cylindrical Bessel function of the first kind.
- [_Tp __K_deriv](#)
The derivative of the modified cylindrical Bessel function of the second kind.
- [_Tp __K_value](#)
The value of the modified cylindrical Bessel function of the second kind.
- [_Tnu __nu_arg](#)
The real order of the modified cylindrical Bessel functions.
- [_Tx __x_arg](#)
The argument of the modified cylindrical Bessel functions.

10.9.1 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 468 of file `specfun_state.h`.

10.9.2 Member Function Documentation

10.9.2.1 __Wronskian()

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__Wronskian ( ) const [inline]
```

Return the Wronskian of this modified cylindrical Bessel function state.

Definition at line 494 of file `specfun_state.h`.

10.9.3 Member Data Documentation

10.9.3.1 `__I_deriv`

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__I_deriv
```

The derivative of the modified cylindrical Bessel function of the first kind.

Definition at line 482 of file `specfun_state.h`.

10.9.3.2 `__I_value`

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__I_value
```

The value of the modified cylindrical Bessel function of the first kind.

Definition at line 478 of file `specfun_state.h`.

10.9.3.3 `__K_deriv`

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__K_deriv
```

The derivative of the modified cylindrical Bessel function of the second kind.

Definition at line 490 of file `specfun_state.h`.

10.9.3.4 `__K_value`

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__K_value
```

The value of the modified cylindrical Bessel function of the second kind.

Definition at line 486 of file `specfun_state.h`.

10.9.3.5 __nu_arg

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tnu __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__nu_arg
```

The real order of the modified cylindrical Bessel functions.

Definition at line 471 of file `specfun_state.h`.

10.9.3.6 __x_arg

```
template<typename _Tnu , typename _Tx , typename _Tp >
_Tx __gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >::__x_arg
```

The argument of the modified cylindrical Bessel functions.

Definition at line 474 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.10 __gnu_cxx::__fock_airy_t< _Tx, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- [_Tp __Wronskian](#) () const
Return the Wronskian of this Fock-type Airy function state.

Public Attributes

- [_Tp __w1_deriv](#)
The derivative of the Fock-type Airy function w1.
- [_Tp __w1_value](#)
The value of the Fock-type Airy function w1.
- [_Tp __w2_deriv](#)
The derivative of the Fock-type Airy function w2.
- [_Tp __w2_value](#)
The value of the Fock-type Airy function w2.
- [_Tx __x_arg](#)
The argument of the Fock-type Airy fuctions.

10.10.1 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 378 of file specfun_state.h.

10.10.2 Member Function Documentation

10.10.2.1 __Wronskian()

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__fock_airy_t<_Tx, _Tp>::__Wronskian ( ) const [inline]
```

Return the Wronskian of this Fock-type Airy function state.

Definition at line 396 of file specfun_state.h.

10.10.3 Member Data Documentation

10.10.3.1 __w1_deriv

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__fock_airy_t<_Tx, _Tp>::__w1_deriv
```

The derivative of the Fock-type Airy function w1.

Definition at line 387 of file specfun_state.h.

10.10.3.2 __w1_value

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__fock_airy_t<_Tx, _Tp>::__w1_value
```

The value of the Fock-type Airy function w1.

Definition at line 384 of file specfun_state.h.

10.10.3.3 __w2_deriv

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__fock_airy_t< _Tx, _Tp >::__w2_deriv
```

The derivative of the Fock-type Airy function w2.

Definition at line 393 of file specfun_state.h.

10.10.3.4 __w2_value

```
template<typename _Tx , typename _Tp >
_Tp __gnu_cxx::__fock_airy_t< _Tx, _Tp >::__w2_value
```

The value of the Fock-type Airy function w2.

Definition at line 390 of file specfun_state.h.

10.10.3.5 __x_arg

```
template<typename _Tx , typename _Tp >
_Tx __gnu_cxx::__fock_airy_t< _Tx, _Tp >::__x_arg
```

The argument of the Fock-type Airy fuctions.

Definition at line 381 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.11 __gnu_cxx::__fp_is_integer_t Struct Reference

```
#include <math_util.h>
```

Public Member Functions

- [operator bool](#) () const
- [operator\(\)](#) () const

Public Attributes

- bool [__is_integral](#)
- int [__value](#)

10.11.1 Detailed Description

A struct returned by floating point integer queries.

Definition at line 123 of file math_util.h.

10.11.2 Member Function Documentation

10.11.2.1 operator bool()

```
__gnu_cxx::__fp_is_integer_t::operator bool ( ) const [inline]
```

Definition at line 132 of file math_util.h.

References [__is_integral](#).

10.11.2.2 operator()()

```
int __gnu_cxx::__fp_is_integer_t::operator() ( ) const [inline]
```

Definition at line 137 of file math_util.h.

References [__value](#).

10.11.3 Member Data Documentation

10.11.3.1 __is_integral

```
bool __gnu_cxx::__fp_is_integer_t::__is_integral
```

Definition at line 126 of file math_util.h.

Referenced by [operator bool\(\)](#).

10.11.3.2 __value

```
int __gnu_cxx::__fp_is_integer_t::__value
```

Definition at line 129 of file `math_util.h`.

Referenced by `operator()()`.

The documentation for this struct was generated from the following file:

- [include/ext/math_util.h](#)

10.12 __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.

10.12.1 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 641 of file `specfun_state.h`.

10.12.2 Member Data Documentation

10.12.2.1 __lgamma_value

```
template<typename _Tp>
_Tp __gnu_cxx::__gamma_inc_t<_Tp>::__lgamma_value
```

The value of the log of the incomplete gamma function.

Definition at line 646 of file `specfun_state.h`.

10.12.2.2 __tgamma_value

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_inc_t<_Tp>::__tgamma_value
```

The value of the total gamma function.

Definition at line 644 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.13 __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.

10.13.1 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 669 of file specfun_state.h.

10.13.2 Member Data Documentation

10.13.2.1 __gamma_1_value

```
template<typename _Tp>
_Tp __gnu_cxx::__gamma_temme_t<_Tp>::__gamma_1_value
```

The output function $\Gamma_1(\mu)$.

Definition at line 681 of file specfun_state.h.

10.13.2.2 __gamma_2_value

```
template<typename _Tp>
_Tp __gnu_cxx::__gamma_temme_t<_Tp>::__gamma_2_value
```

The output function $\Gamma_2(\mu)$.

Definition at line 684 of file specfun_state.h.

10.13.2.3 `__gamma_minus_value`

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_temme_t< _Tp >::__gamma_minus_value
```

The output function $1/\Gamma(1 - \mu)$.

Definition at line 678 of file `specfun_state.h`.

10.13.2.4 `__gamma_plus_value`

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_temme_t< _Tp >::__gamma_plus_value
```

The output function $1/\Gamma(1 + \mu)$.

Definition at line 675 of file `specfun_state.h`.

10.13.2.5 `__mu_arg`

```
template<typename _Tp >
_Tp __gnu_cxx::__gamma_temme_t< _Tp >::__mu_arg
```

The input parameter of the gamma functions.

Definition at line 672 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.14 `__gnu_cxx::__gappa_pq_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Attributes

- `_Tp __gappa_p_value`
- `_Tp __gappa_q_value`

10.14.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__gappa_pq_t< _Tp >
```

Definition at line 614 of file specfun_state.h.

10.14.2 Member Data Documentation

10.14.2.1 __gappa_p_value

```
template<typename _Tp >
_Tp __gnu_cxx::__gappa_pq_t< _Tp >::__gappa_p_value
```

Definition at line 617 of file specfun_state.h.

10.14.2.2 __gappa_q_value

```
template<typename _Tp >
_Tp __gnu_cxx::__gappa_pq_t< _Tp >::__gappa_q_value
```

Definition at line 620 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.15 __gnu_cxx::__gegenbauer_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- [_Tp deriv](#) () const

Public Attributes

- [_Tp __C_n](#)
- [_Tp __C_nm1](#)
- [_Tp __C_nm2](#)
- [_Tp __lambda](#)
- [std::size_t __n](#)
- [_Tp __x](#)

10.15.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__gegenbauer_t<_Tp>
```

A struct to store the state of a Gegenbauer polynomial.

Definition at line 178 of file specfun_state.h.

10.15.2 Member Function Documentation

10.15.2.1 deriv()

```
template<typename _Tp>
_Tp __gnu_cxx::__gegenbauer_t<_Tp>::deriv ( ) const [inline]
```

Definition at line 188 of file specfun_state.h.

10.15.3 Member Data Documentation

10.15.3.1 __C_n

```
template<typename _Tp>
_Tp __gnu_cxx::__gegenbauer_t<_Tp>::__C_n
```

Definition at line 183 of file specfun_state.h.

10.15.3.2 `__C_nm1`

```
template<typename _Tp >  
_Tp \_\_gnu\_cxx::\_\_gegenbauer\_t< _Tp >::__C_nm1
```

Definition at line 184 of file `specfun_state.h`.

10.15.3.3 `__C_nm2`

```
template<typename _Tp >  
_Tp \_\_gnu\_cxx::\_\_gegenbauer\_t< _Tp >::__C_nm2
```

Definition at line 185 of file `specfun_state.h`.

10.15.3.4 `__lambda`

```
template<typename _Tp >  
_Tp \_\_gnu\_cxx::\_\_gegenbauer\_t< _Tp >::__lambda
```

Definition at line 181 of file `specfun_state.h`.

10.15.3.5 `__n`

```
template<typename _Tp >  
std::size_t \_\_gnu\_cxx::\_\_gegenbauer\_t< _Tp >::__n
```

Definition at line 180 of file `specfun_state.h`.

10.15.3.6 `__x`

```
template<typename _Tp >  
_Tp \_\_gnu\_cxx::\_\_gegenbauer\_t< _Tp >::__x
```

Definition at line 182 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.16 __gnu_cxx::__hermite_he_t<_Tp> Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- [_Tp deriv](#) () const
- [_Tp deriv2](#) () const

Public Attributes

- [_Tp __He_n](#)
- [_Tp __He_nm1](#)
- [_Tp __He_nm2](#)
- [std::size_t __n](#)
- [_Tp __x](#)

10.16.1 Detailed Description

```
template<typename _Tp>  
struct __gnu_cxx::__hermite_he_t<_Tp>
```

A struct to store the state of a probabilists Hermite polynomial.

Definition at line 97 of file specfun_state.h.

10.16.2 Member Function Documentation

10.16.2.1 deriv()

```
template<typename _Tp>  
_Tp __gnu_cxx::__hermite_he_t<_Tp>::deriv ( ) const [inline]
```

Definition at line 106 of file specfun_state.h.

10.16.2.2 deriv2()

```
template<typename _Tp>  
_Tp __gnu_cxx::__hermite_he_t<_Tp>::deriv2 ( ) const [inline]
```

Definition at line 110 of file specfun_state.h.

10.16.3 Member Data Documentation

10.16.3.1 __He_n

```
template<typename _Tp >  
_Tp __gnu_cxx::__hermite_he_t< _Tp >::__He_n
```

Definition at line 101 of file specfun_state.h.

10.16.3.2 __He_nm1

```
template<typename _Tp >  
_Tp __gnu_cxx::__hermite_he_t< _Tp >::__He_nm1
```

Definition at line 102 of file specfun_state.h.

10.16.3.3 __He_nm2

```
template<typename _Tp >  
_Tp __gnu_cxx::__hermite_he_t< _Tp >::__He_nm2
```

Definition at line 103 of file specfun_state.h.

10.16.3.4 __n

```
template<typename _Tp >  
std::size_t __gnu_cxx::__hermite_he_t< _Tp >::__n
```

Definition at line 99 of file specfun_state.h.

10.16.3.5 __x

```
template<typename _Tp >
__Tp __gnu_cxx::__hermite_he_t<_Tp>::__x
```

Definition at line 100 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.17 __gnu_cxx::__hermite_t<_Tp> Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- [_Tp deriv](#) () const
- [_Tp deriv2](#) () const

Public Attributes

- [_Tp __H_n](#)
- [_Tp __H_nm1](#)
- [_Tp __H_nm2](#)
- [std::size_t __n](#)
- [_Tp __x](#)

10.17.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__hermite_t<_Tp>
```

A struct to store the state of a Hermite polynomial.

Definition at line 76 of file specfun_state.h.

10.17.2 Member Function Documentation

10.17.2.1 deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::deriv ( ) const [inline]
```

Definition at line 85 of file specfun_state.h.

10.17.2.2 deriv2()

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::deriv2 ( ) const [inline]
```

Definition at line 89 of file specfun_state.h.

10.17.3 Member Data Documentation

10.17.3.1 __H_n

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::__H_n
```

Definition at line 80 of file specfun_state.h.

10.17.3.2 __H_nm1

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::__H_nm1
```

Definition at line 81 of file specfun_state.h.

10.17.3.3 __H_nm2

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::__H_nm2
```

Definition at line 82 of file specfun_state.h.

10.17.3.4 `__n`

```
template<typename _Tp >
std::size_t __gnu_cxx::__hermite_t< _Tp >::__n
```

Definition at line 78 of file `specfun_state.h`.

10.17.3.5 `__x`

```
template<typename _Tp >
_Tp __gnu_cxx::__hermite_t< _Tp >::__x
```

Definition at line 79 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- `include/bits/specfun_state.h`

10.18 `__gnu_cxx::__jacobi_ellint_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __am () const`
- `_Tp __cd () const`
- `_Tp __cn_deriv () 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`
- `_Tp __sn_deriv () const`

Public Attributes

- `_Tp __cn_value`
Jacobi cosine amplitude value.
- `_Tp __dn_value`
Jacobi delta amplitude value.
- `_Tp __sn_value`
Jacobi sine amplitude value.

10.18.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__jacobi_ellint_t< _Tp >
```

Slots for Jacobi elliptic function tuple.

Definition at line 303 of file specfun_state.h.

10.18.2 Member Function Documentation

10.18.2.1 __am()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__am ( ) const [inline]
```

Definition at line 314 of file specfun_state.h.

10.18.2.2 __cd()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__cd ( ) const [inline]
```

Definition at line 332 of file specfun_state.h.

10.18.2.3 __cn_deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__cn_deriv ( ) const [inline]
```

Definition at line 347 of file specfun_state.h.

10.18.2.4 __cs()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__cs ( ) const [inline]
```

Definition at line 335 of file specfun_state.h.

10.18.2.5 __dc()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t<_Tp>::__dc ( ) const [inline]
```

Definition at line 341 of file specfun_state.h.

10.18.2.6 __ds()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t<_Tp>::__ds ( ) const [inline]
```

Definition at line 338 of file specfun_state.h.

10.18.2.7 __nc()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t<_Tp>::__nc ( ) const [inline]
```

Definition at line 320 of file specfun_state.h.

10.18.2.8 __nd()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t<_Tp>::__nd ( ) const [inline]
```

Definition at line 323 of file specfun_state.h.

10.18.2.9 __ns()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t<_Tp>::__ns ( ) const [inline]
```

Definition at line 317 of file specfun_state.h.

10.18.2.10 __sc()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__sc ( ) const [inline]
```

Definition at line 326 of file specfun_state.h.

10.18.2.11 __sd()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__sd ( ) const [inline]
```

Definition at line 329 of file specfun_state.h.

10.18.2.12 __sn_deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__sn_deriv ( ) const [inline]
```

Definition at line 344 of file specfun_state.h.

10.18.3 Member Data Documentation

10.18.3.1 __cn_value

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__cn_value
```

Jacobi cosine amplitude value.

Definition at line 309 of file specfun_state.h.

10.18.3.2 __dn_value

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__dn_value
```

Jacobi delta amplitude value.

Definition at line 312 of file specfun_state.h.

10.18.3.3 __sn_value

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_ellint_t< _Tp >::__sn_value
```

Jacobi sine amplitude value.

Definition at line 306 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.19 __gnu_cxx::__jacobi_t<_Tp> Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- [_Tp deriv](#) () const

Public Attributes

- [_Tp __alpha1](#)
- [_Tp __beta1](#)
- [std::size_t __n](#)
- [_Tp __P_n](#)
- [_Tp __P_nm1](#)
- [_Tp __P_nm2](#)
- [_Tp __x](#)

10.19.1 Detailed Description

```
template<typename _Tp>  
struct __gnu_cxx::__jacobi_t< _Tp >
```

A struct to store the state of a Jacobi polynomial.

Definition at line 154 of file specfun_state.h.

10.19.2 Member Function Documentation

10.19.2.1 deriv()

```
template<typename _Tp >  
_Tp __gnu_cxx::__jacobi_t< _Tp >::deriv ( ) const [inline]
```

Definition at line 165 of file specfun_state.h.

10.19.3 Member Data Documentation

10.19.3.1 __alpha1

```
template<typename _Tp >  
_Tp __gnu_cxx::__jacobi_t< _Tp >::__alpha1
```

Definition at line 157 of file specfun_state.h.

10.19.3.2 __beta1

```
template<typename _Tp >  
_Tp __gnu_cxx::__jacobi_t< _Tp >::__beta1
```

Definition at line 158 of file specfun_state.h.

10.19.3.3 __n

```
template<typename _Tp >
std::size_t __gnu_cxx::__jacobi_t<_Tp>::__n
```

Definition at line 156 of file specfun_state.h.

10.19.3.4 __P_n

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t<_Tp>::__P_n
```

Definition at line 160 of file specfun_state.h.

10.19.3.5 __P_nm1

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t<_Tp>::__P_nm1
```

Definition at line 161 of file specfun_state.h.

10.19.3.6 __P_nm2

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t<_Tp>::__P_nm2
```

Definition at line 162 of file specfun_state.h.

10.19.3.7 __x

```
template<typename _Tp >
_Tp __gnu_cxx::__jacobi_t<_Tp>::__x
```

Definition at line 159 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.20 __gnu_cxx::__laguerre_t<_Tpa, _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- [_Tp deriv](#) () const

Public Attributes

- [_Tpa __alpha1](#)
- [_Tp __L_n](#)
- [_Tp __L_nm1](#)
- [_Tp __L_nm2](#)
- [std::size_t __n](#)
- [_Tp __x](#)

10.20.1 Detailed Description

```
template<typename _Tpa, typename _Tp>  
struct __gnu_cxx::__laguerre_t<_Tpa, _Tp >
```

A struct to store the state of a Laguerre polynomial.

Definition at line 136 of file specfun_state.h.

10.20.2 Member Function Documentation

10.20.2.1 deriv()

```
template<typename _Tpa , typename _Tp >  
_Tp __gnu_cxx::__laguerre_t<_Tpa, _Tp >::deriv ( ) const [inline]
```

Definition at line 146 of file specfun_state.h.

10.20.3 Member Data Documentation

10.20.3.1 `__alpha1`

```
template<typename _Tpa , typename _Tp >  
_Tpa __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__alpha1
```

Definition at line 139 of file `specfun_state.h`.

10.20.3.2 `__L_n`

```
template<typename _Tpa , typename _Tp >  
_Tp __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__L_n
```

Definition at line 141 of file `specfun_state.h`.

10.20.3.3 `__L_nm1`

```
template<typename _Tpa , typename _Tp >  
_Tp __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__L_nm1
```

Definition at line 142 of file `specfun_state.h`.

10.20.3.4 `__L_nm2`

```
template<typename _Tpa , typename _Tp >  
_Tp __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__L_nm2
```

Definition at line 143 of file `specfun_state.h`.

10.20.3.5 `__n`

```
template<typename _Tpa , typename _Tp >  
std::size_t __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__n
```

Definition at line 138 of file `specfun_state.h`.

10.20.3.6 __x

```
template<typename _Tpa , typename _Tp >
_Tp __gnu_cxx::__laguerre_t< _Tpa, _Tp >::__x
```

Definition at line 140 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.21 __gnu_cxx::__legendre_p_t< _Tp > Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- [_Tp deriv](#) () const

Public Attributes

- [std::size_t __l](#)
- [_Tp __P_l](#)
- [_Tp __P_lm1](#)
- [_Tp __P_lm2](#)
- [_Tp __z](#)

10.21.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__legendre_p_t< _Tp >
```

A struct to store the state of a Legendre polynomial.

Definition at line 118 of file specfun_state.h.

10.21.2 Member Function Documentation

10.21.2.1 deriv()

```
template<typename _Tp >
_Tp __gnu_cxx::__legendre_p_t<_Tp >::deriv ( ) const [inline]
```

Definition at line 128 of file specfun_state.h.

10.21.3 Member Data Documentation

10.21.3.1 __l

```
template<typename _Tp >
std::size_t __gnu_cxx::__legendre_p_t<_Tp >::__l
```

Definition at line 120 of file specfun_state.h.

10.21.3.2 __P_l

```
template<typename _Tp >
_Tp __gnu_cxx::__legendre_p_t<_Tp >::__P_l
```

Definition at line 122 of file specfun_state.h.

10.21.3.3 __P_lm1

```
template<typename _Tp >
_Tp __gnu_cxx::__legendre_p_t<_Tp >::__P_lm1
```

Definition at line 123 of file specfun_state.h.

10.21.3.4 __P_lm2

```
template<typename _Tp >
_Tp __gnu_cxx::__legendre_p_t<_Tp >::__P_lm2
```

Definition at line 124 of file specfun_state.h.

10.21.3.5 `__z`

```
template<typename _Tp >
__Tp __gnu_cxx::__legendre_p_t< _Tp >::__z
```

Definition at line 121 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.22 `__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.

10.22.1 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 628 of file `specfun_state.h`.

10.22.2 Member Data Documentation

10.22.2.1 `__lgamma_sign`

```
template<typename _Tp >
int __gnu_cxx::__lgamma_t< _Tp >::__lgamma_sign
```

The sign of the exponent of the log gamma value.

Definition at line 634 of file `specfun_state.h`.

10.22.2.2 `__lgamma_value`

```
template<typename _Tp >
_Tp __gnu_cxx::__lgamma_t<_Tp>::__lgamma_value
```

The value log gamma function.

Definition at line 631 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- `include/bits/specfun_state.h`

10.23 `__gnu_cxx::__quadrature_point_t<_Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `__quadrature_point_t()`=default
- `__quadrature_point_t`(`_Tp __pt`, `_Tp __wt`)

Public Attributes

- `_Tp __point`
- `_Tp __weight`

10.23.1 Detailed Description

```
template<typename _Tp>
struct __gnu_cxx::__quadrature_point_t<_Tp>
```

A structure to store quadrature rules.

Definition at line 59 of file `specfun_state.h`.

10.23.2 Constructor & Destructor Documentation

10.23.2.1 `__quadrature_point_t()` [1/2]

```
template<typename _Tp >
__gnu_cxx::__quadrature_point_t< _Tp >::__quadrature_point_t ( ) [default]
```

10.23.2.2 `__quadrature_point_t()` [2/2]

```
template<typename _Tp >
__gnu_cxx::__quadrature_point_t< _Tp >::__quadrature_point_t (
    _Tp __pt,
    _Tp __wt ) [inline]
```

Definition at line 66 of file `specfun_state.h`.

10.23.3 Member Data Documentation**10.23.3.1** `__point`

```
template<typename _Tp >
_Tp __gnu_cxx::__quadrature_point_t< _Tp >::__point
```

Definition at line 61 of file `specfun_state.h`.

10.23.3.2 `__weight`

```
template<typename _Tp >
_Tp __gnu_cxx::__quadrature_point_t< _Tp >::__weight
```

Definition at line 62 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.24 `__gnu_cxx::__sincos_t< _Tp >` Struct Template Reference

```
#include <specfun_state.h>
```

Public Attributes

- `_Tp __cos_v`
- `_Tp __sin_v`

10.24.1 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 293 of file `specfun_state.h`.

10.24.2 Member Data Documentation

10.24.2.1 `__cos_v`

```
template<typename _Tp>
_Tp __gnu_cxx::__sincos_t<_Tp>::__cos_v
```

Definition at line 296 of file `specfun_state.h`.

Referenced by `std::__detail::__polar_pi()`, and `std::__detail::__sincos_pi()`.

10.24.2.2 `__sin_v`

```
template<typename _Tp>
_Tp __gnu_cxx::__sincos_t<_Tp>::__sin_v
```

Definition at line 295 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:

- `include/bits/specfun_state.h`

10.25 `__gnu_cxx::__sph_bessel_t<_Tn, _Tx, _Tp >` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of this spherical Bessel function state.

Public Attributes

- `_Tp __j_deriv`
The derivative of the spherical Bessel function of the first kind.
- `_Tp __j_value`
The value of the spherical Bessel function of the first kind.
- `_Tn __n_arg`
The integral order of the spherical Bessel functions.
- `_Tp __n_deriv`
The derivative of the spherical Bessel function of the second kind.
- `_Tp __n_value`
The value of the spherical Bessel function of the second kind.
- `_Tx __x_arg`
The argument of the spherical Bessel functions.

10.25.1 Detailed Description

```
template<typename _Tn, typename _Tx, typename _Tp>
struct __gnu_cxx::__sph_bessel_t<_Tn, _Tx, _Tp >
```

Definition at line 528 of file `specfun_state.h`.

10.25.2 Member Function Documentation

10.25.2.1 `__Wronskian()`

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_bessel_t<_Tn, _Tx, _Tp >::__Wronskian ( ) const [inline]
```

Return the Wronskian of this spherical Bessel function state.

Definition at line 549 of file `specfun_state.h`.

10.25.3 Member Data Documentation

10.25.3.1 `__j_deriv`

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__j_deriv
```

The derivative of the spherical Bessel function of the first kind.

Definition at line 540 of file `specfun_state.h`.

10.25.3.2 `__j_value`

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__j_value
```

The value of the spherical Bessel function of the first kind.

Definition at line 537 of file `specfun_state.h`.

10.25.3.3 `__n_arg`

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tn __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__n_arg
```

The integral order of the spherical Bessel functions.

Definition at line 531 of file `specfun_state.h`.

10.25.3.4 `__n_deriv`

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__n_deriv
```

The derivative of the spherical Bessel function of the second kind.

Definition at line 546 of file `specfun_state.h`.

10.25.3.5 `__n_value`

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__n_value
```

The value of the spherical Bessel function of the second kind.

Definition at line 543 of file `specfun_state.h`.

10.25.3.6 `__x_arg`

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tx __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >::__x_arg
```

The argument of the spherical Bessel functions.

Definition at line 534 of file `specfun_state.h`.

The documentation for this struct was generated from the following file:

- `include/bits/specfun_state.h`

10.26 `__gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of this cylindrical Hankel function state.

Public Attributes

- `_Tp __h1_deriv`
The derivative of the spherical Hankel function of the first kind.
- `_Tp __h1_value`
The 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.

10.26.1 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 588 of file `specfun_state.h`.

10.26.2 Member Function Documentation

10.26.2.1 `__Wronskian()`

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_hankel_t<_Tn, _Tx, _Tp>::__Wronskian ( ) const [inline]
```

Return the Wronskian of this cylindrical Hankel function state.

Definition at line 609 of file `specfun_state.h`.

10.26.3 Member Data Documentation

10.26.3.1 `__h1_deriv`

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_hankel_t<_Tn, _Tx, _Tp>::__h1_deriv
```

The derivative of the spherical Hankel function of the first kind.

Definition at line 600 of file `specfun_state.h`.

10.26.3.2 `__h1_value`

```
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 597 of file `specfun_state.h`.

10.26.3.3 __h2_deriv

```
template<typename _Tn , typename _Tx , typename _Tp >  
_Tp __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__h2_deriv
```

The derivative of the spherical Hankel function of the second kind.

Definition at line 606 of file specfun_state.h.

10.26.3.4 __h2_value

```
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 603 of file specfun_state.h.

10.26.3.5 __n_arg

```
template<typename _Tn , typename _Tx , typename _Tp >  
_Tn __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__n_arg
```

The integral order of the spherical Hankel functions.

Definition at line 591 of file specfun_state.h.

10.26.3.6 __x_arg

```
template<typename _Tn , typename _Tx , typename _Tp >  
_Tx __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >::__x_arg
```

The argument of the spherical Hankel functions.

Definition at line 594 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.27 `__gnu_cxx::__sph_mod_bessel_t<_Tn, _Tx, _Tp>` Struct Template Reference

```
#include <specfun_state.h>
```

Public Member Functions

- `_Tp __Wronskian () const`
Return the Wronskian of this modified cylindrical Bessel function state.

Public Attributes

- `_Tp __i_deriv`
The derivative of the modified spherical Bessel function of the first kind.
- `_Tp __i_value`
The value of the modified spherical Bessel function of the first kind.
- `_Tp __k_deriv`
The derivative of the modified spherical Bessel function of the second kind.
- `_Tp __k_value`
The value of the modified spherical Bessel function of the second kind.
- `_Tn __n_arg`
The integral order of the modified spherical Bessel functions.
- `_Tx __x_arg`
The argument of the modified spherical Bessel functions.

10.27.1 Detailed Description

```
template<typename _Tn, typename _Tx, typename _Tp>
struct __gnu_cxx::__sph_mod_bessel_t<_Tn, _Tx, _Tp>
```

Definition at line 554 of file `specfun_state.h`.

10.27.2 Member Function Documentation

10.27.2.1 `__Wronskian()`

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tp __gnu_cxx::__sph_mod_bessel_t<_Tn, _Tx, _Tp>::__Wronskian ( ) const [inline]
```

Return the Wronskian of this modified cylindrical Bessel function state.

Definition at line 580 of file `specfun_state.h`.

10.27.3 Member Data Documentation

10.27.3.1 __i_deriv

```
template<typename _Tn , typename _Tx , typename _Tp >  
_Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__i_deriv
```

The derivative of the modified spherical Bessel function of the first kind.

Definition at line 568 of file specfun_state.h.

10.27.3.2 __i_value

```
template<typename _Tn , typename _Tx , typename _Tp >  
_Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__i_value
```

The value of the modified spherical Bessel function of the first kind.

Definition at line 564 of file specfun_state.h.

10.27.3.3 __k_deriv

```
template<typename _Tn , typename _Tx , typename _Tp >  
_Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__k_deriv
```

The derivative of the modified spherical Bessel function of the second kind.

Definition at line 576 of file specfun_state.h.

10.27.3.4 __k_value

```
template<typename _Tn , typename _Tx , typename _Tp >  
_Tp __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__k_value
```

The value of the modified spherical Bessel function of the second kind.

Definition at line 572 of file specfun_state.h.

10.27.3.5 __n_arg

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tn __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__n_arg
```

The integral order of the modified spherical Bessel functions.

Definition at line 560 of file specfun_state.h.

10.27.3.6 __x_arg

```
template<typename _Tn , typename _Tx , typename _Tp >
_Tx __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >::__x_arg
```

The argument of the modified spherical Bessel functions.

Definition at line 557 of file specfun_state.h.

The documentation for this struct was generated from the following file:

- [include/bits/specfun_state.h](#)

10.28 std::__detail::__gamma_lanczos_data< _Tp > Struct Template Reference

10.28.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::__gamma_lanczos_data< _Tp >
```

A struct for Lanczos algorithm Chebyshev arrays of coefficients.

Definition at line 2018 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- [include/bits/sf_gamma.tcc](#)

10.29 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

10.29.1 Detailed Description

```
template<>
struct std::__detail::__gamma_lanczos_data< double >
```

Definition at line 2040 of file sf_gamma.tcc.

10.29.2 Member Data Documentation

10.29.2.1 `_S_cheby`

```
constexpr std::array<double, 10> std::__detail::__gamma_lanczos_data< double >::_S_cheby [static]
```

Initial value:

```
{
    5.557569219204146e+03,
    -4.248114953727554e+03,
    1.881719608233706e+03,
    -4.705537221412237e+02,
    6.325224688788239e+01,
    -4.206901076213398e+00,
    1.202512485324405e-01,
    -1.141081476816908e-03,
    2.055079676210880e-06,
    1.280568540096283e-09,
}
```

Definition at line 2045 of file sf_gamma.tcc.

10.29.2.2 `_S_g`

```
constexpr double std::__detail::__gamma_lanczos_data< double >::_S_g = 9.5 [static]
```

Definition at line 2042 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- [include/bits/sf_gamma.tcc](#)

10.30 `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

10.30.1 Detailed Description

```
template<>
struct std::__detail::__gamma_lanczos_data< float >
```

Definition at line 2023 of file sf_gamma.tcc.

10.30.2 Member Data Documentation

10.30.2.1 _S_cheby

```
constexpr std::array<float, 7> std::__detail::__gamma_lanczos_data< float >::_S_cheby [static]
```

Initial value:

```
{
    3.307139e+02F,
   -2.255998e+02F,
    6.989520e+01F,
   -9.058929e+00F,
    4.110107e-01F,
   -4.150391e-03F,
   -3.417969e-03F,
}
```

Definition at line 2028 of file sf_gamma.tcc.

10.30.2.2 _S_g

```
constexpr float std::__detail::__gamma_lanczos_data< float >::_S_g = 6.5F [static]
```

Definition at line 2025 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- include/bits/sf_gamma.tcc

10.31 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

10.31.1 Detailed Description

```
template<>
struct std::__detail::__gamma_lanczos_data< long double >
```

Definition at line 2060 of file sf_gamma.tcc.

10.31.2 Member Data Documentation

10.31.2.1 _S_cheby

```
constexpr std::array<long double, 11> std::__detail::__gamma_lanczos_data< long double >::_S_cheby [static]
```

Initial value:

```
{
    1.440399692024250728e+04L,
    -1.128006201837065341e+04L,
    5.384108670160999829e+03L,
    -1.536234184127325861e+03L,
    2.528551924697309561e+02L,
    -2.265389090278717887e+01L,
    1.006663776178612579e+00L,
    -1.900805731354182626e-02L,
    1.150508317664389324e-04L,
    -1.208915136885480024e-07L,
    -1.518856151960790157e-10L,
}
```

Definition at line 2065 of file sf_gamma.tcc.

10.31.2.2 _S_g

```
constexpr long double std::__detail::__gamma_lanczos_data< long double >::_S_g = 10.5L [static]
```

Definition at line 2062 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- [include/bits/sf_gamma.tcc](#)

10.32 std::__detail::__gamma_spouge_data< _Tp > Struct Template Reference

10.32.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::__gamma_spouge_data< _Tp >
```

A struct for Spouge algorithm Chebyshev arrays of coefficients.

Definition at line 1792 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- [include/bits/sf_gamma.tcc](#)

10.33 std::__detail::__gamma_spouge_data< double > Struct Template Reference

Static Public Attributes

- static constexpr std::array< double, 18 > [_S_cheby](#)

10.33.1 Detailed Description

```
template<>
struct std::__detail::__gamma_spouge_data< double >
```

Definition at line 1813 of file sf_gamma.tcc.

10.33.2 Member Data Documentation

10.33.2.1 `_S_cheby`

```
constexpr std::array<double, 18> std::__detail::__gamma_spouge_data< double >::_S_cheby [static]
```

Initial value:

```
{
    2.785716565770350e+08,
   -1.693088166941517e+09,
    4.549688586500031e+09,
   -7.121728036151557e+09,
    7.202572947273274e+09,
   -4.935548868770376e+09,
    2.338187776097503e+09,
   -7.678102458920741e+08,
    1.727524819329867e+08,
   -2.595321377008346e+07,
    2.494811203993971e+06,
   -1.437252641338402e+05,
    4.490767356961276e+03,
   -6.505596924745029e+01,
    3.362323142416327e-01,
   -3.817361443986454e-04,
    3.273137866873352e-08,
   -7.642333165976788e-15,
}
```

Definition at line 1817 of file `sf_gamma.tcc`.

The documentation for this struct was generated from the following file:

- [include/bits/sf_gamma.tcc](#)

10.34 `std::__detail::__gamma_spouge_data< float >` Struct Template Reference

Static Public Attributes

- static constexpr `std::array< float, 7 > _S_cheby`

10.34.1 Detailed Description

```
template<>
struct std::__detail::__gamma_spouge_data< float >
```

Definition at line 1797 of file `sf_gamma.tcc`.

10.34.2 Member Data Documentation

10.34.2.1 __S_cheby

```
constexpr std::array<float, 7> std::__detail::__gamma_spouge_data< float >::__S_cheby [static]
```

Initial value:

```
{  
    2.901419e+03F,  
    -5.929168e+03F,  
    4.148274e+03F,  
    -1.164761e+03F,  
    1.174135e+02F,  
    -2.786588e+00F,  
    3.775392e-03F,  
}
```

Definition at line 1801 of file sf_gamma.tcc.

The documentation for this struct was generated from the following file:

- [include/bits/sf_gamma.tcc](#)

10.35 std::__detail::__gamma_spouge_data< long double > Struct Template Reference

Static Public Attributes

- static constexpr std::array< long double, 22 > [__S_cheby](#)

10.35.1 Detailed Description

```
template<>  
struct std::__detail::__gamma_spouge_data< long double >
```

Definition at line 1840 of file sf_gamma.tcc.

10.35.2 Member Data Documentation

10.35.2.1 `_S_cheby`

```
constexpr std::array<long double, 22> std::__detail::__gamma_spouge_data< long double >::_S_cheby [static]
```

Initial value:

```
{
    1.681473171108908244e+10L,
   -1.269150315503303974e+11L,
    4.339449429013039995e+11L,
   -8.893680202692714895e+11L,
    1.218472425867950986e+12L,
   -1.178403473259353616e+12L,
    8.282455311246278274e+11L,
   -4.292112878930625978e+11L,
    1.646988347276488710e+11L,
   -4.661514921989111004e+10L,
    9.619972564515443397e+09L,
   -1.419382551781042824e+09L,
    1.454145470816386107e+08L,
   -9.923020719435758179e+06L,
    4.253557563919127284e+05L,
   -1.053371059784341875e+04L,
    1.332425479537961437e+02L,
   -7.118343974029489132e-01L,
    1.172051640057979518e-03L,
   -3.323940885824119041e-07L,
    4.503801674404338524e-12L,
   -5.320477002211632680e-20L,
}
```

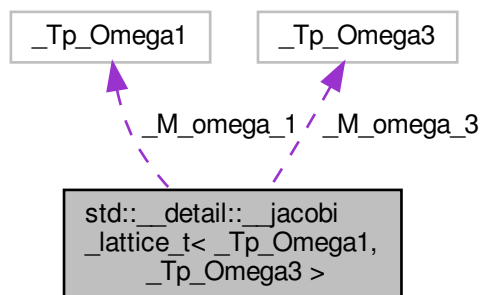
Definition at line 1844 of file `sf_gamma.tcc`.

The documentation for this struct was generated from the following file:

- `include/bits/sf_gamma.tcc`

10.36 `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >` Struct Template Reference

Collaboration diagram for `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >`:



Classes

- struct [__arg_t](#)
- struct [__tau_t](#)

Public Types

- using [_Cmplx](#) = std::complex< [_Real](#) >
- using [_Real](#) = __gnu_cxx::fp_promote_t< [_Real_Omega1](#), [_Real_Omega3](#) >
- using [_Real_Omega1](#) = __num_traits_t< [_Tp_Omega1](#) >
- using [_Real_Omega3](#) = __num_traits_t< [_Tp_Omega3](#) >
- using [_Tp_Nome](#) = std::conditional_t< __gnu_cxx::is_complex_v< [_Tp_Omega1](#) > && __gnu_cxx::is_

Public Member Functions

- [__jacobi_lattice_t](#) (const [_Tp_Omega1](#) &[__omega1](#), const [_Tp_Omega3](#) &[__omega3](#))
Construct the lattice from two complex lattice frequencies.
- [__jacobi_lattice_t](#) (const [__tau_t](#) &[__tau](#))
Construct the lattice from a single complex lattice parameter or half period ratio.
- [__jacobi_lattice_t](#) ([_Tp_Nome](#) [__q](#))
Construct the lattice from a single scalar elliptic nome.
- [_Tp_Nome](#) [__ellnome](#) () const
- [_Tp_Omega1](#) [__omega_1](#) () const
Return the first lattice frequency.
- [_Cmplx](#) [__omega_2](#) () const
Return the second lattice frequency.
- [_Tp_Omega3](#) [__omega_3](#) () const
Return the third lattice frequency.
- [__arg_t](#) [__reduce](#) (const [_Cmplx](#) &[__z](#)) const
- [__tau_t](#) [__tau](#) () const
Return the acalar lattice parameter or half period ratio.

Public Attributes

- [_Tp_Omega1](#) [_M_omega_1](#)
- [_Tp_Omega3](#) [_M_omega_3](#)

Static Public Attributes

- static constexpr auto [_S_pi](#) = __gnu_cxx::__const_pi<[_Real](#)>()

10.36.1 Detailed Description

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
struct std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >
```

A struct representing the Jacobi and Weierstrass lattice. The two types for the frequencies and the subsequent type calculus allow us to treat the rectangular lattice (real nome, pure imaginary lattice parameter) specially.

Definition at line 470 of file sf_theta.tcc.

10.36.2 Member Typedef Documentation

10.36.2.1 _Cmplx

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
using std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__Cmplx = std::complex<_Real>
```

Definition at line 478 of file sf_theta.tcc.

10.36.2.2 _Real

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
using std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__Real = __gnu_cxx::fp_promote←
_t<_Real_Omega1, _Real_Omega3>
```

Definition at line 477 of file sf_theta.tcc.

10.36.2.3 _Real_Omega1

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
using std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__Real_Omega1 = __num_traits←
t<_Tp_Omega1>
```

Definition at line 475 of file sf_theta.tcc.

10.36.2.4 _Real_Omega3

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
using std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__Real_Omega3 = __num_traits_↵
t<_Tp_Omega3>
```

Definition at line 476 of file sf_theta.tcc.

10.36.2.5 _Tp_Nome

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
using std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__Tp_Nome = std::conditional_↵
t<__gnu_cxx::is_complex_v<_Tp_Omega1> && __gnu_cxx::is_complex_v<_Tp_Omega3>, _Cmplx, _Real>
```

Definition at line 481 of file sf_theta.tcc.

10.36.3 Constructor & Destructor Documentation

10.36.3.1 __jacobi_lattice_t() [1/3]

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__jacobi_lattice_t (
    const _Tp_Omega1 & __omega1,
    const _Tp_Omega3 & __omega3 ) [inline]
```

Construct the lattice from two complex lattice frequencies.

Definition at line 508 of file sf_theta.tcc.

10.36.3.2 __jacobi_lattice_t() [2/3]

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__jacobi_lattice_t (
    const __tau_t & __tau ) [inline], [explicit]
```

Construct the lattice from a single complex lattice parameter or half period ratio.

Definition at line 530 of file sf_theta.tcc.

10.36.3.3 `__jacobi_lattice_t()` [3/3]

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__jacobi_lattice_t (
    _Tp_Nome __q ) [inline], [explicit]
```

Construct the lattice from a single scalar elliptic nome.

Definition at line 549 of file `sf_theta.tcc`.

10.36.4 Member Function Documentation

10.36.4.1 `__ellnome()`

```
template<typename _Tp_Omega1 , typename _Tp_Omega3 >
__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__Tp_Nome std::__detail::__jacobi_lattice_t< _Tp_↵
Omega1, _Tp_Omega3 >::__ellnome ( ) const
```

Return the elliptic nome corresponding to the lattice parameter.

Definition at line 593 of file `sf_theta.tcc`.

Referenced by `std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::__jacobi_theta_0_t()`, and `std::__detail::__jacobi_↵
lattice_t< _Tp1, _Tp3 >::__omega_3()`.

10.36.4.2 `__omega_1()`

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
_Tp_Omega1 std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__omega_1 ( ) const [inline]
```

Return the first lattice frequency.

Definition at line 564 of file `sf_theta.tcc`.

Referenced by `std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::__jacobi_theta_0_t()`, and `std::__detail::__↵
weierstrass_roots_t< _Tp1, _Tp3 >::__weierstrass_roots_t()`.

10.36.4.3 __omega_2()

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
_Cmplx std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__omega_2 ( ) const [inline]
```

Return the second lattice frequency.

Definition at line 569 of file sf_theta.tcc.

Referenced by std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::__jacobi_theta_0_t().

10.36.4.4 __omega_3()

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
_Tp_Omega3 std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__omega_3 ( ) const [inline]
```

Return the third lattice frequency.

Definition at line 574 of file sf_theta.tcc.

Referenced by std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::__jacobi_theta_0_t().

10.36.4.5 __reduce()

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
__jacobi_lattice_t< _Tp1, _Tp3 >::__arg_t std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__↵
reduce (
    const _Cmplx & __z ) const
```

Reduce the argument to the fundamental lattice parallelogram $(0, 2\pi, 2\pi(1 + \tau), 2\pi\tau)$. This is sort of like a 2D lattice remquo.

Parameters

__↵	The argument to be reduced.
__z	

Returns

A struct containing the argument reduced to the interior of the fundamental parallelogram and two integers indicating the number of periods in the 'real' and 'tau' directions.

Definition at line 616 of file sf_theta.tcc.

Referenced by `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__ellnome()`, `std::__detail::__jacobi_theta_1()`, `std::__detail::__jacobi_theta_2()`, `std::__detail::__jacobi_theta_3()`, `std::__detail::__jacobi_theta_4()`, and `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__omega_3()`.

10.36.4.6 `__tau()`

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
__tau_t std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau ( ) const [inline]
```

Return the acalar lattice parameter or half period ratio.

Definition at line 559 of file `sf_theta.tcc`.

Referenced by `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__ellnome()`, `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__jacobi_lattice_t()`, `std::__detail::__jacobi_theta_1()`, `std::__detail::__jacobi_theta_2()`, `std::__detail::__jacobi_theta_3()`, `std::__detail::__jacobi_theta_4()`, and `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__reduce()`.

10.36.5 Member Data Documentation

10.36.5.1 `_M_omega_1`

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
_Tp_Omega1 std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::_M_omega_1
```

Definition at line 584 of file `sf_theta.tcc`.

Referenced by `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__jacobi_lattice_t()`, `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__omega_1()`, `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__omega_2()`, and `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__tau()`.

10.36.5.2 `_M_omega_3`

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
_Tp_Omega3 std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::_M_omega_3
```

Definition at line 585 of file `sf_theta.tcc`.

Referenced by `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__jacobi_lattice_t()`, `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__omega_2()`, `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__omega_3()`, and `std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__tau()`.

10.36.5.3 _S_pi

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
constexpr auto std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::_S_pi = __gnu_cxx::__
__const_pi<_Real>() [static]
```

Definition at line 583 of file sf_theta.tcc.

Referenced by std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__ellnome(), std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__jacobi_lattice_t(), std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::__jacobi_theta_0_t(), std::__detail::__jacobi_theta_1_t(), std::__detail::__jacobi_theta_2_t(), std::__detail::__jacobi_theta_3_t(), std::__detail::__jacobi_theta_4_t(), std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__reduce(), and std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__weierstrass_roots_t().

The documentation for this struct was generated from the following file:

- include/bits/sf_theta.tcc

10.37 std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__arg_t Struct Reference

Public Attributes

- int __m
- int __n
- _Cmplx __z

10.37.1 Detailed Description

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
struct std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__arg_t
```

A struct representing a complex argument reduced to the 'central' lattice cell.

Definition at line 500 of file sf_theta.tcc.

10.37.2 Member Data Documentation

10.37.2.1 `__m`

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
int std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__arg_t::__m
```

Definition at line 502 of file `sf_theta.tcc`.

10.37.2.2 `__n`

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
int std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__arg_t::__n
```

Definition at line 503 of file `sf_theta.tcc`.

10.37.2.3 `__z`

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
_Cmplx std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__arg_t::__z
```

Definition at line 504 of file `sf_theta.tcc`.

The documentation for this struct was generated from the following file:

- [include/bits/sf_theta.tcc](#)

10.38 `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau_t` Struct Reference

Public Member Functions

- [__tau_t](#)([_Cmplx](#) [__tau](#))

Public Attributes

- [_Cmplx](#) [__val](#)

10.38.1 Detailed Description

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
struct std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau_t
```

A struct representing a complex scalar lattice parameter or half period ratio.

Definition at line 487 of file sf_theta.tcc.

10.38.2 Constructor & Destructor Documentation

10.38.2.1 __tau_t()

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau_t::__tau_t (
    _Cmplx __tau ) [inline], [explicit]
```

Definition at line 491 of file sf_theta.tcc.

Referenced by std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__jacobi_lattice_t(), and std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__tau_t().

10.38.3 Member Data Documentation

10.38.3.1 __val

```
template<typename _Tp_Omega1, typename _Tp_Omega3 = std::complex<_Tp_Omega1>>
_Cmplx std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau_t::__val
```

Definition at line 489 of file sf_theta.tcc.

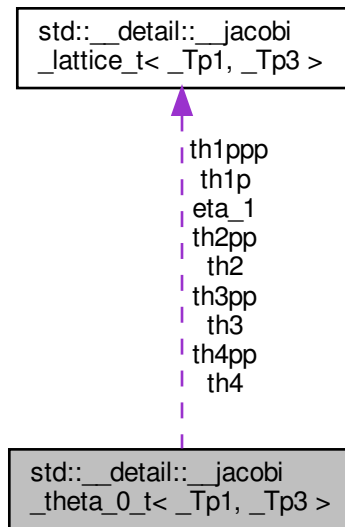
Referenced by std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__ellnome(), std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__jacobi_lattice_t(), and std::__detail::__jacobi_lattice_t< _Tp1, _Tp3 >::__reduce().

The documentation for this struct was generated from the following file:

- [include/bits/sf_theta.tcc](#)

10.39 `std::__detail::__jacobi_theta_0_t<_Tp1, _Tp3 >` Struct Template Reference

Collaboration diagram for `std::__detail::__jacobi_theta_0_t<_Tp1, _Tp3 >`:



Public Types

- using `_Cmplx` = `std::complex<_Real >`
- using `_Real` = `__num_traits_t<_Type >`
- using `_Type` = `typename __jacobi_lattice_t<_Tp1, _Tp3 >::_Tp_Nome`

Public Member Functions

- `__jacobi_theta_0_t` (const `__jacobi_lattice_t<_Tp1, _Tp3 > &__lattice`)
- `_Type dedekind_eta` () const

Public Attributes

- `_Type eta_1`
- `_Cmplx eta_2`
- `_Cmplx eta_3`
- `_Type th1p`
- `_Type th1ppp`
- `_Type th2`
- `_Type th2pp`
- `_Type th3`
- `_Type th3pp`
- `_Type th4`
- `_Type th4pp`

10.39.1 Detailed Description

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
struct std::__detail::__jacobi_theta_0_t<_Tp1, _Tp3 >
```

A struct for the non-zero theta functions and their derivatives at zero argument.

Definition at line 643 of file sf_theta.tcc.

10.39.2 Member Typedef Documentation

10.39.2.1 _Cmplx

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
using std::__detail::__jacobi_theta_0_t<_Tp1, _Tp3 >::__Cmplx = std::complex<_Real>
```

Definition at line 649 of file sf_theta.tcc.

10.39.2.2 _Real

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
using std::__detail::__jacobi_theta_0_t<_Tp1, _Tp3 >::__Real = __num_traits_t<_Type>
```

Definition at line 648 of file sf_theta.tcc.

10.39.2.3 _Type

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
using std::__detail::__jacobi_theta_0_t<_Tp1, _Tp3 >::__Type = typename __jacobi_lattice_t<_Tp1,
_Tp3>::__Tp_None
```

Definition at line 647 of file sf_theta.tcc.

10.39.3 Constructor & Destructor Documentation

10.39.3.1 `__jacobi_theta_0_t()`

```
template<typename _Tp1 , typename _Tp3 >
std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::__jacobi_theta_0_t (
    const __jacobi_lattice_t< _Tp1, _Tp3 > & __lattice )
```

Return a struct of the Jacobi theta functions and up to three non-zero derivatives evaluated at zero argument.

Definition at line 674 of file `sf_theta.tcc`.

References `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__ellnome()`, `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__omega_1()`, `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__omega_2()`, `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__omega_3()`, and `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__S_pi`.

Referenced by `std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::dedekind_eta()`.

10.39.4 Member Function Documentation

10.39.4.1 `dedekind_eta()`

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::dedekind_eta ( ) const [inline]
```

Definition at line 664 of file `sf_theta.tcc`.

References `std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::__jacobi_theta_0_t()`.

10.39.5 Member Data Documentation

10.39.5.1 `eta_1`

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::eta_1
```

Definition at line 659 of file `sf_theta.tcc`.

10.39.5.2 eta_2

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Cmplx std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::eta_2
```

Definition at line 660 of file sf_theta.tcc.

10.39.5.3 eta_3

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Cmplx std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::eta_3
```

Definition at line 661 of file sf_theta.tcc.

10.39.5.4 th1p

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th1p
```

Definition at line 651 of file sf_theta.tcc.

10.39.5.5 th1ppp

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th1ppp
```

Definition at line 652 of file sf_theta.tcc.

10.39.5.6 th2

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th2
```

Definition at line 653 of file sf_theta.tcc.

Referenced by std::__detail::__jacobi_theta_2(), and std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__weierstrass_roots_t().

10.39.5.7 th2pp

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th2pp
```

Definition at line 654 of file sf_theta.tcc.

10.39.5.8 th3

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th3
```

Definition at line 655 of file sf_theta.tcc.

Referenced by std::__detail::__jacobi_theta_3().

10.39.5.9 th3pp

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th3pp
```

Definition at line 656 of file sf_theta.tcc.

10.39.5.10 th4

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th4
```

Definition at line 657 of file sf_theta.tcc.

Referenced by std::__detail::__jacobi_theta_4(), and std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__↔weierstrass_roots_t().

10.39.5.11 th4pp

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th4pp
```

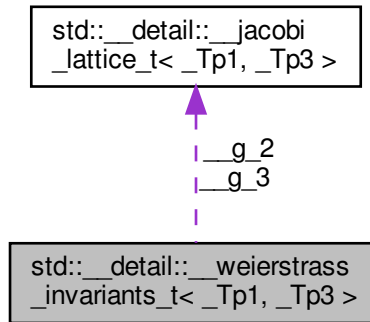
Definition at line 658 of file sf_theta.tcc.

The documentation for this struct was generated from the following file:

- [include/bits/sf_theta.tcc](#)

10.40 std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 > Struct Template Reference

Collaboration diagram for std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >:



Public Types

- using `_Cmplx` = `std::complex< _Real >`
- using `_Real` = `__num_traits_t< _Type >`
- using `_Type` = `typename __jacobi_lattice_t< _Tp1, _Tp3 >::Tp_Nome`

Public Member Functions

- `__weierstrass_invariants_t` (const `__jacobi_lattice_t< _Tp1, _Tp3 > &`)
- `_Type __delta` () const
Return the discriminant $\Delta = g_2^3 - 27g_3^2$.
- `_Type __klein_j` () const
Return Klein's invariant $J = 1738g_2^3/(g_2^3 - 27g_3^2)$.

Public Attributes

- `_Type __g_2`
- `_Type __g_3`

10.40.1 Detailed Description

```
template<typename _Tp1, typename _Tp3>
struct std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >
```

A struct of the Weierstrass elliptic function invariants.

$$g_2 = 2(e_1e_2 + e_2e_3 + e_3e_1)$$

$$g_3 = 4(e_1e_2e_3)$$

Definition at line 826 of file `sf_theta.tcc`.

10.40.2 Member Typedef Documentation

10.40.2.1 `_Cmplx`

```
template<typename _Tp1 , typename _Tp3 >
using std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >::__Cmplx = std::complex<_Real>
```

Definition at line 830 of file sf_theta.tcc.

10.40.2.2 `_Real`

```
template<typename _Tp1 , typename _Tp3 >
using std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >::__Real = __num_traits_t<_Type>
```

Definition at line 829 of file sf_theta.tcc.

10.40.2.3 `_Type`

```
template<typename _Tp1 , typename _Tp3 >
using std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >::__Type = typename __jacobi_lattice←
_t<_Tp1, _Tp3>::__Tp_Nome
```

Definition at line 828 of file sf_theta.tcc.

10.40.3 Constructor & Destructor Documentation

10.40.3.1 `__weierstrass_invariants_t()`

```
template<typename _Tp1 , typename _Tp3 >
std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >::__weierstrass_invariants_t (
    const __jacobi_lattice_t< _Tp1, _Tp3 > & __lattice )
```

Constructor for the Weierstrass invariants.

$$g_2 = 2(e_1e_2 + e_2e_3 + e_3e_1)$$

$$g_3 = 4(e_1e_2e_3)$$

Definition at line 864 of file sf_theta.tcc.

References `std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__e1`.

Referenced by `std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >::__klein_j()`.

10.40.4 Member Function Documentation

10.40.4.1 __delta()

```
template<typename _Tp1 , typename _Tp3 >
_Type std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >::__delta ( ) const [inline]
```

Return the discriminant $\Delta = g_2^3 - 27g_3^2$.

Definition at line 838 of file sf_theta.tcc.

10.40.4.2 __klein_j()

```
template<typename _Tp1 , typename _Tp3 >
_Type std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >::__klein_j ( ) const [inline]
```

Return Klein's invariant $J = 1738g_2^3/(g_2^3 - 27g_3^2)$.

Definition at line 846 of file sf_theta.tcc.

References `std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >::__weierstrass_invariants_t()`.

10.40.5 Member Data Documentation

10.40.5.1 __g_2

```
template<typename _Tp1 , typename _Tp3 >
_Type std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >::__g_2
```

Definition at line 832 of file sf_theta.tcc.

10.40.5.2 __g_3

```
template<typename _Tp1 , typename _Tp3 >
_Type std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >::__g_3
```

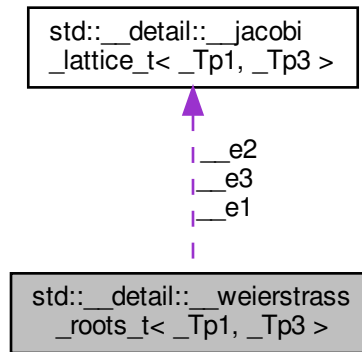
Definition at line 832 of file sf_theta.tcc.

The documentation for this struct was generated from the following file:

- `include/bits/sf_theta.tcc`

10.41 std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 > Struct Template Reference

Collaboration diagram for std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >:



Public Types

- using [_Cmplx](#) = std::complex< [_Real](#) >
- using [_Real](#) = __num_traits_t< [_Type](#) >
- using [_Type](#) = typename [__jacobi_lattice_t](#)< _Tp1, _Tp3 >::__Tp_Nome

Public Member Functions

- [__weierstrass_roots_t](#) (const [__jacobi_lattice_t](#)< _Tp1, _Tp3 > &__lattice)
- [__weierstrass_roots_t](#) (const [__jacobi_theta_0_t](#)< _Tp1, _Tp3 > &__theta0, _Tp1 __omega1)
- [_Type](#) [__delta](#) () const

Return the discriminant $\Delta = 16(e_2 - e_3)^2(e_3 - e_1)^2(e_1 - e_2)^2$.

Public Attributes

- [_Type](#) [__e1](#)
- [_Type](#) [__e2](#)
- [_Type](#) [__e3](#)

10.41.1 Detailed Description

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
struct std::__detail::__weierstrass_roots_t<_Tp1, _Tp3>
```

A struct of the Weierstrass elliptic function roots.

$$e_1 = \frac{\pi^2}{12\omega_1^2}(\theta_2^4(q, 0) + 2\theta_4^4(q, 0))$$

$$e_2 = \frac{\pi^2}{12\omega_1^2}(\theta_2^4(q, 0) - \theta_4^4(q, 0))$$

$$e_3 = \frac{\pi^2}{12\omega_1^2}(-2\theta_2^4(q, 0) - \theta_4^4(q, 0))$$

Note that $e_1 + e_2 + e_3 = 0$

Definition at line 747 of file sf_theta.tcc.

10.41.2 Member Typedef Documentation

10.41.2.1 _Cmplx

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
using std::__detail::__weierstrass_roots_t<_Tp1, _Tp3>::__Cmplx = std::complex<_Real>
```

Definition at line 751 of file sf_theta.tcc.

10.41.2.2 _Real

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
using std::__detail::__weierstrass_roots_t<_Tp1, _Tp3>::__Real = __num_traits_t<_Type>
```

Definition at line 750 of file sf_theta.tcc.

10.41.2.3 _Type

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
using std::__detail::__weierstrass_roots_t<_Tp1, _Tp3>::__Type = typename __jacobi_lattice_t<_Tp1, _Tp3>::__Tp_None
```

Definition at line 749 of file sf_theta.tcc.

10.41.3 Constructor & Destructor Documentation

10.41.3.1 `__weierstrass_roots_t()` [1/2]

```
template<typename _Tp1 , typename _Tp3 >
std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__weierstrass_roots_t (
    const __jacobi_lattice_t< _Tp1, _Tp3 > & __lattice ) [explicit]
```

Constructor for the Weierstrass roots.

Parameters

<code>__lattice</code>	The Jacobi latticece.
------------------------	-----------------------

Definition at line 781 of file `sf_theta.tcc`.

Referenced by `std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__delta()`.

10.41.3.2 `__weierstrass_roots_t()` [2/2]

```
template<typename _Tp1 , typename _Tp3 >
std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__weierstrass_roots_t (
    const __jacobi_theta_0_t< _Tp1, _Tp3 > & __theta0,
    _Tp1 __omega_1 )
```

Constructor for the Weierstrass roots.

Parameters

<code>__lattice</code>	The Jacobi latticece.
------------------------	-----------------------

Definition at line 799 of file `sf_theta.tcc`.

References `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__omega_1()`, `std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__S_pi`, `std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th2`, and `std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >::th4`.

10.41.4 Member Function Documentation

10.41.4.1 __delta()

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__delta ( ) const [inline]
```

Return the discriminant $\Delta = 16(e_2 - e_3)^2(e_3 - e_1)^2(e_1 - e_2)^2$.

Definition at line 764 of file sf_theta.tcc.

References std::__detail::__weierstrass_roots_t<_Tp1, _Tp3>::__weierstrass_roots_t().

10.41.5 Member Data Documentation

10.41.5.1 __e1

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__e1
```

Definition at line 753 of file sf_theta.tcc.

Referenced by std::__detail::__weierstrass_invariants_t<_Tp1, _Tp3>::__weierstrass_invariants_t().

10.41.5.2 __e2

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__e2
```

Definition at line 753 of file sf_theta.tcc.

10.41.5.3 __e3

```
template<typename _Tp1, typename _Tp3 = std::complex<_Tp1>>
_Type std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >::__e3
```

Definition at line 753 of file sf_theta.tcc.

The documentation for this struct was generated from the following file:

- [include/bits/sf_theta.tcc](#)

10.42 std::__detail::_Airy< _Tp > Class Template Reference

Public Types

- using [scalar_type](#) = __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 }

10.42.1 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 2503 of file sf_airy.tcc.

10.42.2 Member Typedef Documentation

10.42.2.1 scalar_type

```
template<typename _Tp>
using std::__detail::_Airy< _Tp >::scalar_type = __num_traits_t<value_type>
```

Definition at line 2508 of file sf_airy.tcc.

10.42.2.2 value_type

```
template<typename _Tp>
using std::__detail::_Airy<_Tp>::value_type = _Tp
```

Definition at line 2507 of file sf_airy.tcc.

10.42.3 Constructor & Destructor Documentation

10.42.3.1 _Airy() [1/3]

```
template<typename _Tp>
constexpr std::__detail::_Airy<_Tp>::_Airy ( ) [default]
```

10.42.3.2 _Airy() [2/3]

```
template<typename _Tp>
std::__detail::_Airy<_Tp>::_Airy (
    const _Airy<_Tp> & ) [default]
```

10.42.3.3 _Airy() [3/3]

```
template<typename _Tp>
std::__detail::_Airy<_Tp>::_Airy (
    _Airy<_Tp> && ) [default]
```

10.42.4 Member Function Documentation

10.42.4.1 operator>()

```
template<typename _Tp>
constexpr _AiryState<_Tp> std::__detail::_Airy<_Tp>::operator() (
    value_type __y ) const
```

Return the Airy functions for complex argument.

Definition at line 2526 of file sf_airy.tcc.

References `std::__detail::__beta()`, `std::__detail::_Airy_series<_Tp>::_S_Ai()`, and `std::__detail::_Airy_series<_Tp>::_S_Bi()`.

10.42.5 Member Data Documentation

10.42.5.1 inner_radius

```
template<typename _Tp>
scalar_type std::__detail::_Airy< _Tp >::inner_radius {__Airy_default_radii<scalar_type>::inner←
_radius}
```

Definition at line 2517 of file sf_airy.tcc.

10.42.5.2 outer_radius

```
template<typename _Tp>
scalar_type std::__detail::_Airy< _Tp >::outer_radius {__Airy_default_radii<scalar_type>::outer←
_radius}
```

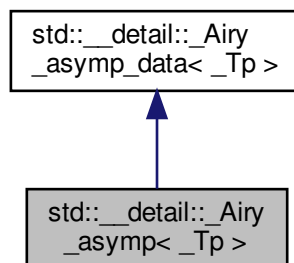
Definition at line 2518 of file sf_airy.tcc.

The documentation for this class was generated from the following file:

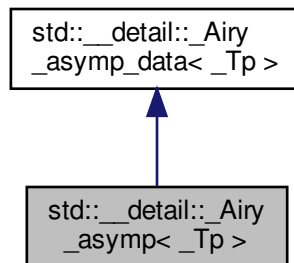
- [include/bits/sf_airy.tcc](#)

10.43 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`

10.43.1 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 1997 of file `sf_airy.tcc`.

10.43.2 Member Typedef Documentation

10.43.2.1 _Cmplx

```
template<typename _Tp >
using std::__detail::_Airy_asymp< _Tp >::_Cmplx = std::complex<_Tp>
```

Definition at line 2002 of file sf_airy.tcc.

10.43.3 Constructor & Destructor Documentation

10.43.3.1 _Airy_asymp()

```
template<typename _Tp >
constexpr std::__detail::_Airy_asymp< _Tp >::_Airy_asymp ( ) [default]
```

10.43.4 Member Function Documentation

10.43.4.1 _S_absarg_ge_pio3()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_asymp< _Tp >::_S_absarg_ge_pio3 (
    _Cmplx __z ) const
```

This function evaluates $Ai(z)$, $Ai'(z)$ and $Bi(z)$, $Bi'(z)$ from their asymptotic expansions for $|arg(z)| < 2 * \pi/3$ i.e. roughly along the negative real axis.

Template Parameters

<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 2270 of file sf_airy.tcc.

References std::__detail::_AiryState<_Tp>::__z.

10.43.4.2 _S_absarg_lt_pio3()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_asymp< _Tp >::_S_absarg_lt_pio3 (
    _Cmplx __z ) const
```

This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(-z)| < \pi/3$ i.e. roughly along the negative real axis.

For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined for $|z|$. This function assumes $|z| > 15$ and $|arg(-z)| < \pi/3$.

Note that for speed and since this function is called by another, checks for valid arguments are not made. Hence, an error return is not needed.

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Parameters

in	<code>__z</code>	The value at which the Airy function and their derivatives are evaluated.
----	------------------	---

Returns

A struct containing z , $Ai(z)$, $Ai'(z)$, $Bi(z)$, $Bi'(z)$.

Todo Revisit these numbers of terms for the Airy asymptotic expansions.

Definition at line 2300 of file sf_airy.tcc.

References std::__detail::_AiryState<_Tp>::__z.

10.43.4.3 operator()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_asymp< _Tp >::operator() (
    _Cmplx __t,
    bool __return_fock_airy = false ) const
```

Return the Airy functions for a given argument using asymptotic series.

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 2028 of file `sf_airy.tcc`.

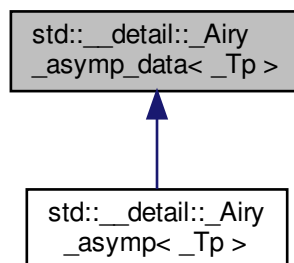
References `std::__detail::_AiryState< _Tp >::__z`.

The documentation for this class was generated from the following file:

- [include/bits/sf_airy.tcc](#)

10.44 std::__detail::_Airy_asymp_data< _Tp > Struct Template Reference

Inheritance diagram for `std::__detail::_Airy_asymp_data< _Tp >`:



10.44.1 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 631 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- [include/bits/sf_airy.tcc](#)

10.45 `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

10.45.1 Detailed Description

```
template<>
struct std::__detail::_Airy_asymp_data< double >
```

Definition at line 738 of file `sf_airy.tcc`.

10.45.2 Member Data Documentation

10.45.2.1 `_S_c`

```
constexpr double std::__detail::_Airy_asymp_data< double >::_S_c[_S_max_cd] [static]
```

Definition at line 744 of file `sf_airy.tcc`.

10.45.2.2 `_S_d`

```
constexpr double std::__detail::_Airy_asymp_data< double >::_S_d[_S_max_cd] [static]
```

Definition at line 947 of file `sf_airy.tcc`.

10.45.2.3 `_S_max_cd`

```
constexpr int std::__detail::_Airy_asymp_data< double >::_S_max_cd = 198 [static]
```

Definition at line 740 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- `include/bits/sf_airy.tcc`

10.46 `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

10.46.1 Detailed Description

```
template<>
struct std::__detail::_Airy_asymp_data< float >
```

Definition at line 635 of file `sf_airy.tcc`.

10.46.2 Member Data Documentation

10.46.2.1 `_S_c`

```
constexpr float std::__detail::_Airy_asymp_data< float >::_S_c[_S_max_cd] [static]
```

Definition at line 641 of file `sf_airy.tcc`.

10.46.2.2 `_S_d`

```
constexpr float std::__detail::_Airy_asymp_data< float >::_S_d[_S_max_cd] [static]
```

Definition at line 689 of file `sf_airy.tcc`.

10.46.2.3 _S_max_cd

```
constexpr int std::__detail::_Airy_asymp_data< float >::_S_max_cd = 43 [static]
```

Definition at line 637 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- include/bits/sf_airy.tcc

10.47 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

10.47.1 Detailed Description

```
template<>
struct std::__detail::_Airy_asymp_data< long double >
```

Definition at line 1151 of file sf_airy.tcc.

10.47.2 Member Data Documentation

10.47.2.1 _S_c

```
constexpr long double std::__detail::_Airy_asymp_data< long double >::_S_c[\_S\_max\_cd] [static]
```

Definition at line 1157 of file sf_airy.tcc.

10.47.2.2 _S_d

```
constexpr long double std::__detail::_Airy_asymp_data< long double >::_S_d[\_S\_max\_cd] [static]
```

Definition at line 1363 of file sf_airy.tcc.

10.47.2.3 `_S_max_cd`

```
constexpr int std::__detail::_Airy_asymp_data< long double >::_S_max_cd = 201 [static]
```

Definition at line 1153 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- `include/bits/sf_airy.tcc`

10.48 `std::__detail::_Airy_asymp_series< _Sum >` Class Template Reference

Public Types

- using `scalar_type` = `__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{})`

10.48.1 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 2363 of file `sf_airy.tcc`.

10.48.2 Member Typedef Documentation

10.48.2.1 scalar_type

```
template<typename _Sum>
using std::__detail::_Airy_asymp_series<_Sum>::scalar_type = __num_traits_t<value_type>
```

Definition at line 2368 of file sf_airy.tcc.

10.48.2.2 value_type

```
template<typename _Sum>
using std::__detail::_Airy_asymp_series<_Sum>::value_type = typename _Sum::value_type
```

Definition at line 2367 of file sf_airy.tcc.

10.48.3 Constructor & Destructor Documentation

10.48.3.1 _Airy_asymp_series() [1/3]

```
template<typename _Sum>
std::__detail::_Airy_asymp_series<_Sum>::_Airy_asymp_series (
    _Sum __proto ) [inline]
```

Definition at line 2372 of file sf_airy.tcc.

10.48.3.2 _Airy_asymp_series() [2/3]

```
template<typename _Sum>
std::__detail::_Airy_asymp_series<_Sum>::_Airy_asymp_series (
    const _Airy_asymp_series<_Sum> & ) [default]
```

10.48.3.3 `_Airy_asymp_series()` [3/3]

```
template<typename _Sum>
std::__detail::_Airy_asymp_series< _Sum >::_Airy_asymp_series (
    _Airy_asymp_series< _Sum > && ) [default]
```

10.48.4 Member Function Documentation

10.48.4.1 `operator()()`

```
template<typename _Sum>
_AiryState< typename _Airy_asymp_series< _Sum >::value_type > std::__detail::_Airy_asymp_series<
_Sum >::operator() (
    value_type __y )
```

Return an `__AiryState` containing, not actual Airy functions, but four asymptotic Airy components:

Template Parameters

<code>_Sum</code>	A sum type
-------------------	------------

Definition at line 2417 of file `sf_airy.tcc`.

10.48.5 Member Data Documentation

10.48.5.1 `_S_sqrt_pi`

```
template<typename _Sum>
constexpr _Airy_asymp_series< _Sum >::scalar_type std::__detail::_Airy_asymp_series< _Sum >::↔
S_sqrt_pi = __gnu_cxx::__const_root_pi(scalar_type{}) [static]
```

Definition at line 2370 of file `sf_airy.tcc`.

The documentation for this class was generated from the following file:

- `include/bits/sf_airy.tcc`

10.49 std::__detail::_Airy_default_radii< _Tp > Struct Template Reference

10.49.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_Airy_default_radii< _Tp >
```

Definition at line 2474 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- [include/bits/sf_airy.tcc](#)

10.50 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}

10.50.1 Detailed Description

```
template<>
struct std::__detail::_Airy_default_radii< double >
```

Definition at line 2485 of file sf_airy.tcc.

10.50.2 Member Data Documentation

10.50.2.1 inner_radius

```
constexpr double std::\_\_detail::\_Airy\_default\_radii< double >::inner\_radius {4.0} [static]
```

Definition at line 2487 of file sf_airy.tcc.

10.50.2.2 outer_radius

```
constexpr double std::__detail::_Airy_default_radii< double >::outer_radius {12.0} [static]
```

Definition at line 2488 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- [include/bits/sf_airy.tcc](#)

10.51 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}

10.51.1 Detailed Description

```
template<>
struct std::__detail::_Airy_default_radii< float >
```

Definition at line 2478 of file sf_airy.tcc.

10.51.2 Member Data Documentation

10.51.2.1 inner_radius

```
constexpr float std::__detail::_Airy_default_radii< float >::inner_radius {2.0F} [static]
```

Definition at line 2480 of file sf_airy.tcc.

10.51.2.2 outer_radius

```
constexpr float std::__detail::_Airy_default_radii< float >::outer_radius {6.0F} [static]
```

Definition at line 2481 of file sf_airy.tcc.

The documentation for this struct was generated from the following file:

- [include/bits/sf_airy.tcc](#)

10.52 `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}

10.52.1 Detailed Description

```
template<>
struct std::__detail::_Airy_default_radii< long double >
```

Definition at line 2492 of file `sf_airy.tcc`.

10.52.2 Member Data Documentation

10.52.2.1 `inner_radius`

```
constexpr long double std::__detail::_Airy_default_radii< long double >::inner_radius {5.0L}
[static]
```

Definition at line 2494 of file `sf_airy.tcc`.

10.52.2.2 `outer_radius`

```
constexpr long double std::__detail::_Airy_default_radii< long double >::outer_radius {15.0L}
[static]
```

Definition at line 2495 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- `include/bits/sf_airy.tcc`

10.53 `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](#){})

10.53.1 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 107 of file sf_airy.tcc.

10.53.2 Member Typedef Documentation

10.53.2.1 _Cmplx

```
template<typename _Tp >
using std::__detail::_Airy_series< _Tp >::_Cmplx = std::complex<_Tp>
```

Definition at line 111 of file sf_airy.tcc.

10.53.3 Member Function Documentation

10.53.3.1 _S_Ai()

```
template<typename _Tp >
std::pair< std::complex< _Tp >, std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Ai
(
    _Cmplx __t ) [static]
```

Return the Airy function of the first kind and its derivative by using the series expansions of the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

The Airy function of the first kind is then defined by:

$$Ai(x) = Ai(0)fai(x) + Ai'(0)gai(x)$$

where $Ai(0) = 3^{-2/3}/\Gamma(2/3)$, $Ai'(0) = -3^{-1/2}Bi'(0)$ and $Bi(0) = 3^{1/2}Ai(0)$, $Bi'(0) = 3^{1/6}/\Gamma(1/3)$

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 340 of file sf_airy.tcc.

Referenced by std::__detail::_Airy<_Tp>::operator()().

10.53.3.2 _S_Airy()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Airy (
    _Cmplx __t ) [static]
```

Return the Fock-type Airy functions $Ai(t)$, and $Bi(t)$ and their derivatives of complex argument.

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Parameters

<code>←</code>	The complex argument
<code>←</code>	
<code>←</code>	
<code>←</code>	
<code>t</code>	

Definition at line 608 of file `sf_airy.tcc`.

10.53.3.3 `_S_Bi()`

```
template<typename _Tp >
std::pair< std::complex< _Tp >, std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Bi
(
    _Cmplx __t ) [static]
```

Return the Airy function of the second kind and its derivative by using the series expansions of the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

The Airy function of the second kind is then defined by:

$$Bi(x) = Bi(0)fai(x) + Bi'(0)gai(x)$$

where $Ai(0) = 3^{-2/3}/\Gamma(2/3)$, $Ai'(0) = -3-1/2Bi'(0)$ and $Bi(0) = 3^{1/2}Ai(0)$, $Bi'(0) = 3^{1/6}/\Gamma(1/3)$

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 363 of file `sf_airy.tcc`.

Referenced by `std::__detail::_Airy< _Tp >::operator()()`.

10.53.3.4 _S_FGH()

```
template<typename _Tp >
_AiryAuxilliaryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_FGH (
    _Cmplx __t ) [static]
```

Return the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$hai(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Definition at line 382 of file sf_airy.tcc.

10.53.3.5 _S_Fock()

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Fock (
    _Cmplx __t ) [static]
```

Return the Fock-type Airy functions $w_1(t)$, and $w_2(t)$ and their derivatives of complex argument.

Template Parameters

<code>_Tp</code>	A real type
------------------	-------------

Parameters

\leftarrow	The complex argument
\leftarrow	
\leftarrow	
\leftarrow	
t	

Definition at line 620 of file sf_airy.tcc.

10.53.3.6 `_S_Scorer()`

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Scorer (
    _Cmplx __t ) [static]
```

Return the Scorer functions by using the series expansions of the auxilliary Airy functions:

$$fai(x) = \sum_{k=0}^{\infty} \frac{(2k+1)!!!x^{3k}}{(2k+1)!}$$

$$gai(x) = \sum_{k=0}^{\infty} \frac{(2k+2)!!!x^{3k+1}}{(2k+2)!}$$

$$hai(x) = \sum_{k=0}^{\infty} \frac{(2k+3)!!!x^{3k+2}}{(2k+3)!}$$

The Scorer function is then defined by:

$$Hi(x) = Hi(0) (fai(x) + gai(x) + hai(x))$$

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 463 of file `sf_airy.tcc`.

10.53.3.7 `_S_Scorer2()`

```
template<typename _Tp >
_AiryState< std::complex< _Tp > > std::__detail::_Airy_series< _Tp >::_S_Scorer2 (
    _Cmplx __t ) [static]
```

Return the Scorer functions by using the series expansions:

$$Hi(x) = \frac{3^{-2/3}}{\pi} \sum_{k=0}^{\infty} \Gamma\left(\frac{k+1}{3}\right) \frac{3^{1/3}x}{k!}$$

$$Hi'(x) = \frac{3^{-1/3}}{\pi} \sum_{k=0}^{\infty} \Gamma\left(\frac{k+2}{3}\right) \frac{3^{1/3}x}{k!}$$

$$Gi(x) = \frac{3^{-2/3}}{\pi} \sum_{k=0}^{\infty} \cos\left(\frac{2k-1}{3}\pi\right) \Gamma\left(\frac{k+1}{3}\right) \frac{3^{1/3}x}{k!}$$

$$Gi'(x) = \frac{3^{-1/3}}{\pi} \sum_{k=0}^{\infty} \cos\left(\frac{2k+1}{3}\pi\right) \Gamma\left(\frac{k+2}{3}\right) \frac{3^{1/3}x}{k!}$$

Definition at line 500 of file sf_airy.tcc.

References std::__detail::__gamma().

10.53.4 Member Data Documentation

10.53.4.1 _N_FGH

```
template<typename _Tp >
constexpr int std::__detail::_Airy_series<_Tp>::_N_FGH = 200 [static]
```

Definition at line 113 of file sf_airy.tcc.

10.53.4.2 _S_Ai0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Ai0 = _Tp{3.550280538878172392600631860041831763980e-1↵
L} [static]
```

Definition at line 129 of file sf_airy.tcc.

10.53.4.3 _S_Aip0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series<_Tp>::_S_Aip0 = _Tp{-2.588194037928067984051835601892039634793e-1↵
L} [static]
```

Definition at line 131 of file sf_airy.tcc.

10.53.4.4 `_S_Bi0`

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Bi0 = _Tp{6.149266274460007351509223690936135535960e-1↵
L} [static]
```

Definition at line 133 of file `sf_airy.tcc`.

10.53.4.5 `_S_Bip0`

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Bip0 = _Tp{4.482883573538263579148237103988283908668e-1↵
L} [static]
```

Definition at line 135 of file `sf_airy.tcc`.

10.53.4.6 `_S_eps`

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_eps = __gnu_cxx::__epsilon(_Tp{}) [static]
```

Definition at line 124 of file `sf_airy.tcc`.

10.53.4.7 `_S_Gi0`

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Gi0 = _Tp{2.049755424820002450503074563645378511979e-1↵
L} [static]
```

Definition at line 141 of file `sf_airy.tcc`.

10.53.4.8 `_S_Gip0`

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Gip0 = _Tp{1.494294524512754526382745701329427969551e-1↵
L} [static]
```

Definition at line 143 of file `sf_airy.tcc`.

10.53.4.9 _S_Hi0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Hi0 = _Tp{4.099510849640004901006149127290757023959e-1↵
L} [static]
```

Definition at line 137 of file sf_airy.tcc.

10.53.4.10 _S_Hip0

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_Hip0 = _Tp{2.988589049025509052765491402658855939102e-1↵
L} [static]
```

Definition at line 139 of file sf_airy.tcc.

10.53.4.11 _S_i

```
template<typename _Tp >
constexpr std::complex< _Tp > std::__detail::_Airy_series< _Tp >::_S_i {_Tp{0}, _Tp{1}} [static]
```

Definition at line 144 of file sf_airy.tcc.

10.53.4.12 _S_pi

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_pi = __gnu_cxx::__const_pi(_Tp{}) [static]
```

Definition at line 125 of file sf_airy.tcc.

10.53.4.13 _S_sqrt_pi

```
template<typename _Tp >
constexpr _Tp std::__detail::_Airy_series< _Tp >::_S_sqrt_pi = __gnu_cxx::__const_root_pi(_Tp{})
[static]
```

Definition at line 127 of file sf_airy.tcc.

The documentation for this class was generated from the following file:

- [include/bits/sf_airy.tcc](#)

10.54 std::__detail::_AiryAuxilliaryState< _Tp > Struct Template Reference

Public Types

- using [_Val](#) = __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](#)

10.54.1 Detailed Description

```
template<typename _Tp>  
struct std::__detail::_AiryAuxilliaryState< _Tp >
```

A structure containing three auxilliary Airy functions and their derivatives.

Definition at line 79 of file sf_airy.tcc.

10.54.2 Member Typedef Documentation

10.54.2.1 _Val

```
template<typename _Tp>  
using std::\_\_detail::\_AiryAuxilliaryState< \_Tp >::\_Val = __num_traits_t<_Tp>
```

Definition at line 81 of file sf_airy.tcc.

10.54.3 Member Data Documentation

10.54.3.1 __fai_deriv

```
template<typename _Tp>
_Tp std::__detail::__AiryAuxilliaryState< _Tp >::__fai_deriv
```

Definition at line 85 of file sf_airy.tcc.

10.54.3.2 __fai_value

```
template<typename _Tp>
_Tp std::__detail::__AiryAuxilliaryState< _Tp >::__fai_value
```

Definition at line 84 of file sf_airy.tcc.

10.54.3.3 __gai_deriv

```
template<typename _Tp>
_Tp std::__detail::__AiryAuxilliaryState< _Tp >::__gai_deriv
```

Definition at line 87 of file sf_airy.tcc.

10.54.3.4 __gai_value

```
template<typename _Tp>
_Tp std::__detail::__AiryAuxilliaryState< _Tp >::__gai_value
```

Definition at line 86 of file sf_airy.tcc.

10.54.3.5 __hai_deriv

```
template<typename _Tp>
_Tp std::__detail::__AiryAuxilliaryState< _Tp >::__hai_deriv
```

Definition at line 89 of file sf_airy.tcc.

10.54.3.6 `__hai_value`

```
template<typename _Tp>
_Tp std::__detail::_AiryAuxilliaryState< _Tp >::__hai_value
```

Definition at line 88 of file `sf_airy.tcc`.

10.54.3.7 `__z`

```
template<typename _Tp>
_Tp std::__detail::_AiryAuxilliaryState< _Tp >::__z
```

Definition at line 83 of file `sf_airy.tcc`.

The documentation for this struct was generated from the following file:

- `include/bits/sf_airy.tcc`

10.55 `std::__detail::_AiryState< _Tp >` Struct Template Reference

Public Types

- using `_Real` = `__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`

10.55.1 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 54 of file sf_airy.tcc.

10.55.2 Member Typedef Documentation

10.55.2.1 _Real

```
template<typename _Tp>
using std::__detail::_AiryState<_Tp>::_Real = __num_traits_t<_Tp>
```

Definition at line 56 of file sf_airy.tcc.

10.55.3 Member Function Documentation

10.55.3.1 true_Wronskian()

```
template<typename _Tp>
_Real std::__detail::_AiryState<_Tp>::true_Wronskian ( ) [inline]
```

Definition at line 69 of file sf_airy.tcc.

10.55.3.2 Wronskian()

```
template<typename _Tp>
_Tp std::__detail::_AiryState<_Tp>::Wronskian ( ) const [inline]
```

Definition at line 65 of file sf_airy.tcc.

References `std::__detail::_AiryState<_Tp>::_Ai_deriv`.

10.55.4 Member Data Documentation

10.55.4.1 `__Ai_deriv`

```
template<typename _Tp>
_Tp std::__detail::__AiryState< _Tp >::__Ai_deriv
```

Definition at line 60 of file `sf_airy.tcc`.

Referenced by `std::__detail::__AiryState< _Tp >::Wronskian()`.

10.55.4.2 `__Ai_value`

```
template<typename _Tp>
_Tp std::__detail::__AiryState< _Tp >::__Ai_value
```

Definition at line 59 of file `sf_airy.tcc`.

10.55.4.3 `__Bi_deriv`

```
template<typename _Tp>
_Tp std::__detail::__AiryState< _Tp >::__Bi_deriv
```

Definition at line 62 of file `sf_airy.tcc`.

10.55.4.4 `__Bi_value`

```
template<typename _Tp>
_Tp std::__detail::__AiryState< _Tp >::__Bi_value
```

Definition at line 61 of file `sf_airy.tcc`.

10.55.4.5 __z

```
template<typename _Tp>
_Tp std::__detail::_AiryState< _Tp >::__z
```

Definition at line 58 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:

- include/bits/sf_airy.tcc

10.56 std::__detail::_AsympTerminator< _Tp > Class Template Reference

Public Member Functions

- [_AsympTerminator](#) (std::size_t __max_iter, _Real __mul=_Real{1})
- std::size_t [num_terms](#) () const
Return the current number of terms summed.
- bool [operator\(\)](#) (_Tp __term, _Tp __sum)
Detect if the sum should terminate either because the incoming term is small enough or because the terms are starting to grow or.
- _Tp [operator<<](#) (_Tp __term)
Filter a term before applying it to the sum.

10.56.1 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 107 of file sf_polylog.tcc.

10.56.2 Constructor & Destructor Documentation

10.56.2.1 `_AsympTerminator()`

```
template<typename _Tp>
std::__detail::_AsympTerminator< _Tp >::_AsympTerminator (
    std::size_t __max_iter,
    _Real __mul = _Real{1} ) [inline]
```

Definition at line 120 of file `sf_polylog.tcc`.

10.56.3 Member Function Documentation

10.56.3.1 `num_terms()`

```
template<typename _Tp>
std::size_t std::__detail::_AsympTerminator< _Tp >::num_terms ( ) const [inline]
```

Return the current number of terms summed.

Definition at line 140 of file `sf_polylog.tcc`.

10.56.3.2 `operator>()`

```
template<typename _Tp>
bool std::__detail::_AsympTerminator< _Tp >::operator() (
    _Tp __term,
    _Tp __sum ) [inline]
```

Detect if the sum should terminate either because the incoming term is small enough or because the terms are starting to grow or.

Definition at line 147 of file `sf_polylog.tcc`.

10.56.3.3 `operator<<()`

```
template<typename _Tp>
_Tp std::__detail::_AsympTerminator< _Tp >::operator<< (
    _Tp __term ) [inline]
```

Filter a term before applying it to the sum.

Definition at line 127 of file `sf_polylog.tcc`.

The documentation for this class was generated from the following file:

- [include/bits/sf_polylog.tcc](#)

10.57 std::__detail::_Factorial_table< _Tp > Struct Template Reference

Public Attributes

- [_Tp __factorial](#)
- [_Tp __log_factorial](#)
- [int __n](#)

10.57.1 Detailed Description

```
template<typename _Tp>
struct std::__detail::_Factorial_table< _Tp >
```

Definition at line 67 of file sf_gamma.tcc.

10.57.2 Member Data Documentation

10.57.2.1 __factorial

```
template<typename _Tp >
_Tp std::__detail::_Factorial_table< _Tp >::__factorial
```

Definition at line 70 of file sf_gamma.tcc.

Referenced by `std::__detail::__double_factorial()`, and `std::__detail::__gamma_reciprocal()`.

10.57.2.2 __log_factorial

```
template<typename _Tp >
_Tp std::__detail::_Factorial_table< _Tp >::__log_factorial
```

Definition at line 71 of file sf_gamma.tcc.

Referenced by `std::__detail::__log_double_factorial()`, and `std::__detail::__log_gamma()`.

10.57.2.3 `__n`

```
template<typename _Tp >
int std::__detail::_Factorial_table< _Tp >::__n
```

Definition at line 69 of file `sf_gamma.tcc`.

Referenced by `std::__detail::__binomial()`, `std::__detail::__digamma()`, `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::__polygamma()`, and `std::__detail::__rising_factorial()`.

The documentation for this struct was generated from the following file:

- `include/bits/sf_gamma.tcc`

10.58 `std::__detail::_Terminator< _Tp >` Class Template Reference

Public Member Functions

- `_Terminator` (`std::size_t __max_iter`, `_Real __mul=_Real{1}`)
- `std::size_t num_terms ()` const
Return the current number of terms summed.
- `bool operator() (_Tp __term, _Tp __sum)`
Detect if the sum should terminate either because the incoming term is small enough or the maximum number of terms has been reached.

10.58.1 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 62 of file `sf_polylog.tcc`.

10.58.2 Constructor & Destructor Documentation

10.58.2.1 _Terminator()

```
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.

10.58.3 Member Function Documentation

10.58.3.1 num_terms()

```
template<typename _Tp>
std::size_t std::__detail::_Terminator< _Tp >::num_terms ( ) const [inline]
```

Return the current number of terms summed.

Definition at line 80 of file sf_polylog.tcc.

10.58.3.2 operator>()

```
template<typename _Tp>
bool std::__detail::_Terminator< _Tp >::operator() (
    _Tp __term,
    _Tp __sum ) [inline]
```

Detect if the sum should terminate either because the incoming term is small enough or the maximum number of terms has been reached.

Definition at line 86 of file sf_polylog.tcc.

The documentation for this class was generated from the following file:

- [include/bits/sf_polylog.tcc](#)

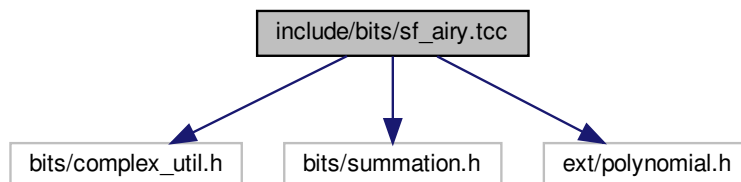
Chapter 11

File Documentation

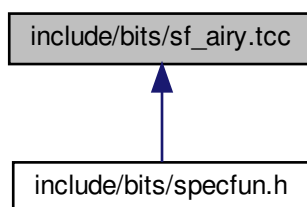
11.1 include/bits/sf_airy.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_AIRY_TCC 1](#)

Functions

- [template<typename _Tp>](#)
[std::complex<_Tp>](#) [std::__detail::__airy_ai](#) ([std::complex<_Tp>](#) __z)
Return the complex Airy Ai function.
- [template<typename _Tp>](#)
[std::complex<_Tp>](#) [std::__detail::__airy_bi](#) ([std::complex<_Tp>](#) __z)
Return the complex Airy Bi function.

Variables

- [template<typename _Tp>](#)
[constexpr int](#) [std::__detail::__max_FGH](#) = [_Airy_series<_Tp>::_N_FGH](#)
- [template<>](#)
[constexpr int](#) [std::__detail::__max_FGH<double>](#) = 79
- [template<>](#)
[constexpr int](#) [std::__detail::__max_FGH<float>](#) = 15

11.1.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.1.2 Macro Definition Documentation

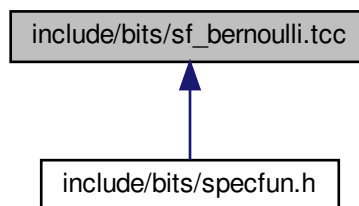
11.1.2.1 _GLIBCXX_BITS_SF_AIRY_TCC

```
#define _GLIBCXX_BITS_SF_AIRY_TCC 1
```

Definition at line 31 of file sf_airy.tcc.

11.2 include/bits/sf_bernoulli.tcc File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_BERNOULLI_TCC 1](#)

Functions

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli` (unsigned int __n)
This returns Bernoulli number B_n .
- `template<typename _Tp >`
`_Tp std::__detail::__bernoulli` (unsigned int __n, _Tp __x)
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n` (unsigned int __n)
This returns Bernoulli number B_{2n} at even integer arguments $2n$.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series` (unsigned int __n)
This returns Bernoulli numbers from a table or by summation for larger values.

$$B_{2n} = (-1)^{n+1} 2 \frac{(2n)!}{(2\pi)^{2n}} \zeta(2n)$$

11.2.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.2.2 Macro Definition Documentation

11.2.2.1 `_GLIBCXX_BITS_SF_BERNOULLI_TCC`

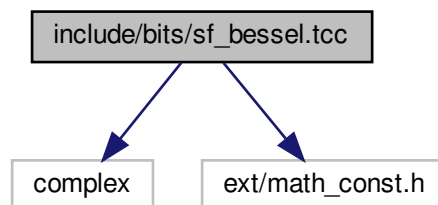
```
#define _GLIBCXX_BITS_SF_BERNOULLI_TCC 1
```

Definition at line 35 of file `sf_bernoulli.tcc`.

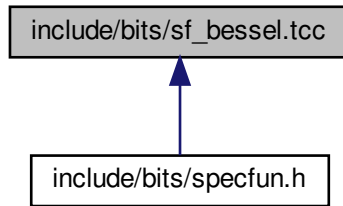
11.3 `include/bits/sf_bessel.tcc` File Reference

```
#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](#)

Implementation-space details.

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 __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_steeds](#) ([_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 _GLIBCXX_BITS_SF_BESSEL_TCC

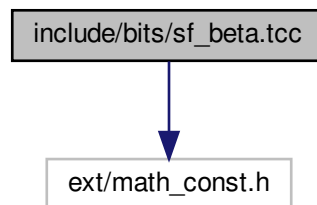
```
#define _GLIBCXX_BITS_SF_BESSEL_TCC 1
```

Definition at line 47 of file sf_bessel.tcc.

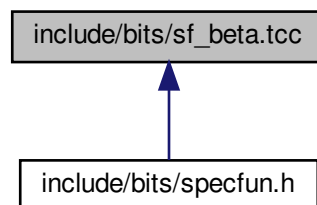
11.4 include/bits/sf_beta.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_BETA_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__beta (_Tp __a, _Tp __b)`
Return the beta function $B(a, b)$.
- `template<typename _Tp >`
`_Tp std::__detail::__beta_gamma (_Tp __a, _Tp __b)`
Return the beta function: $B(a, b)$.
- `template<typename _Tp >`
`_Tp std::__detail::__beta_inc (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__beta_lgamma (_Tp __a, _Tp __b)`
Return the beta function $B(a, b)$ using the log gamma functions.
- `template<typename _Tp >`
`_Tp std::__detail::__beta_product (_Tp __a, _Tp __b)`
Return the beta function $B(x, y)$ using the product form.
- `template<typename _Tp >`
`_Tp std::__detail::__ibeta_cont_frac (_Tp __a, _Tp __b, _Tp __x)`

11.4.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.4.2 Macro Definition Documentation

11.4.2.1 [_GLIBCXX_BITS_SF_BETA_TCC](#)

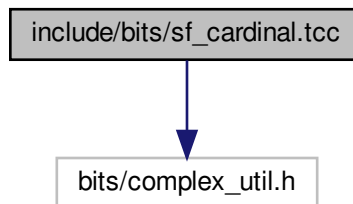
```
#define _GLIBCXX_BITS_SF_BETA_TCC 1
```

Definition at line 49 of file `sf_beta.tcc`.

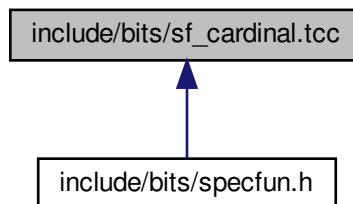
11.5 include/bits/sf_cardinal.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_CARDINAL_TCC](#) 1

Functions

- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > std::__detail::__sinc (_Tp __x)`

Return the sinus cardinal function

$$\text{sinc}(x) = \frac{\sin(x)}{x}$$

- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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 _GLIBCXX_BITS_SF_CARDINAL_TCC

```
#define _GLIBCXX_BITS_SF_CARDINAL_TCC 1
```

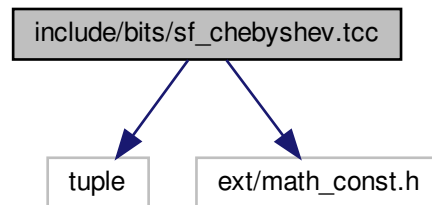
Definition at line 31 of file sf_cardinal.tcc.

11.6 include/bits/sf_chebyshev.tcc File Reference

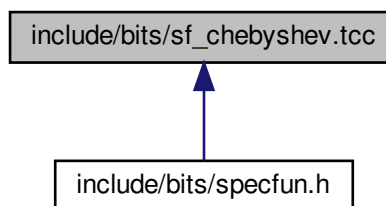
```
#include <tuple>
```

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_CHEBYSHEV_TCC 1](#)

Functions

- `template<typename _Tp >`
`std::tuple< _Tp, _Tp, _Tp > std::__detail::__chebyshev_recur (unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1)`
- `template<typename _Tp >`
`__gnu_cxx::__chebyshev_t_t< _Tp > std::__detail::__chebyshev_t (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__chebyshev_u_t< _Tp > std::__detail::__chebyshev_u (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__chebyshev_v_t< _Tp > std::__detail::__chebyshev_v (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__chebyshev_w_t< _Tp > std::__detail::__chebyshev_w (unsigned int __n, _Tp __x)`

11.6.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.6.2 Macro Definition Documentation

11.6.2.1 `_GLIBCXX_BITS_SF_CHEBYSHEV_TCC`

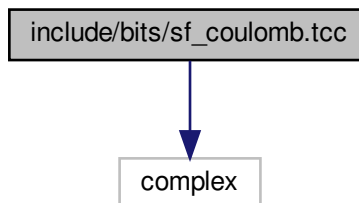
```
#define _GLIBCXX_BITS_SF_CHEBYSHEV_TCC 1
```

Definition at line 31 of file `sf_chebyshev.tcc`.

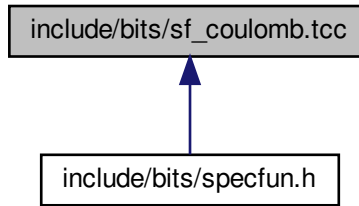
11.7 `include/bits/sf_coulomb.tcc` File Reference

```
#include <complex>
```

Include dependency graph for `sf_coulomb.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_COULOMB_TCC 1](#)

Functions

- [template<typename _Tp > std::pair< _Tp, _Tp > std::__detail::__coulomb_CF1 \(unsigned int __l, _Tp __eta, _Tp __x\)](#)
- [template<typename _Tp > std::complex< _Tp > std::__detail::__coulomb_CF2 \(unsigned int __l, _Tp __eta, _Tp __x\)](#)
- [template<typename _Tp > std::pair< _Tp, _Tp > std::__detail::__coulomb_f_recur \(unsigned int __l_min, unsigned int __k_max, _Tp __eta, _Tp __x, _Tp _F_l_max, _Tp _Fp_l_max\)](#)
- [template<typename _Tp > std::pair< _Tp, _Tp > std::__detail::__coulomb_g_recur \(unsigned int __l_min, unsigned int __k_max, _Tp __eta, _Tp __x, _Tp _G_l_min, _Tp _Gp_l_min\)](#)
- [template<typename _Tp > _Tp std::__detail::__coulomb_norm \(unsigned int __l, _Tp __eta\)](#)
- [template<typename _Tp > std::complex< _Tp > std::__detail::__hydrogen \(unsigned int __n, unsigned int __l, unsigned int __m, _Tp __Z, _Tp __r, _Tp __theta, _Tp __phi\)](#)

11.7.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.7.2 Macro Definition Documentation

11.7.2.1 _GLIBCXX_BITS_SF_COULOMB_TCC

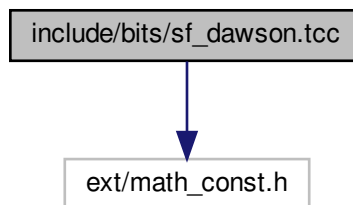
```
#define _GLIBCXX_BITS_SF_COULOMB_TCC 1
```

Definition at line 31 of file sf_coulomb.tcc.

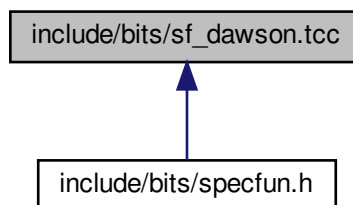
11.8 include/bits/sf_dawson.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- `#define _GLIBCXX_BITS_SF_DAWSON_TCC 1`

Functions

- `template<typename _Tp >
_Tp std::__detail::__dawson (_Tp __x)`
Return the Dawson integral, $F(x)$, for real argument x .
- `template<typename _Tp >
_Tp std::__detail::__dawson_cont_frac (_Tp __x)`
Compute the Dawson integral using a sampling theorem representation.
- `template<typename _Tp >
_Tp std::__detail::__dawson_series (_Tp __x)`
Compute the Dawson integral using the series expansion.

11.8.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.8.2 Macro Definition Documentation

11.8.2.1 _GLIBCXX_BITS_SF_DAWSON_TCC

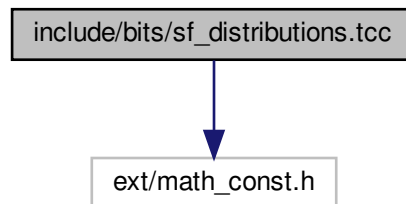
```
#define _GLIBCXX_BITS_SF_DAWSON_TCC 1
```

Definition at line 31 of file `sf_dawson.tcc`.

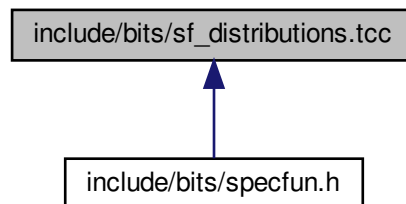
11.9 include/bits/sf_distributions.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC](#) 1

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__beta_p (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__binomial_p (_Tp __p, unsigned int __n, unsigned int __k)`
Return the 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::__binomial_q (_Tp __p, unsigned int __n, unsigned int __k)`
Return the complementary binomial cumulative distribution function.
- `template<typename _Tp >`
`_Tp std::__detail::__cauchy_p (_Tp __a, _Tp __b, _Tp __x)`
- `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_p (_Tp __lambda, _Tp __x)`
Return the exponential cumulative probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__exponential_pdf (_Tp __lambda, _Tp __x)`
Return the exponential probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__exponential_q (_Tp __lambda, _Tp __x)`
Return the complement of the exponential cumulative probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__fisher_f_p (_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::__fisher_f_q (_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_p (_Tp __alpha, _Tp __beta, _Tp __x)`
Return the gamma 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::__gamma_q (_Tp __alpha, _Tp __beta, _Tp __x)`
Return the gamma complementary cumulative propability distribution function.
- `template<typename _Tp >`
`_Tp std::__detail::__kolmogorov_p (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__logistic_p (_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_p (_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_p (_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_p (_Tp __t, unsigned int __nu)`
Return 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::__student_t_q (_Tp __t, unsigned int __nu)`
Return the complement of the Students T probability function.
- `template<typename _Tp >`
`_Tp std::__detail::__weibull_p (_Tp __a, _Tp __b, _Tp __x)`
Return the Weibull cumulative probability density function.
- `template<typename _Tp >`
`_Tp std::__detail::__weibull_pdf (_Tp __a, _Tp __b, _Tp __x)`
Return the Weibull probability density function.

11.9.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.9.2 Macro Definition Documentation

11.9.2.1 _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC

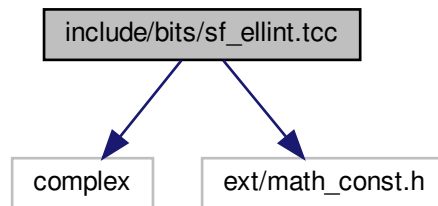
```
#define _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC 1
```

Definition at line 49 of file sf_distributions.tcc.

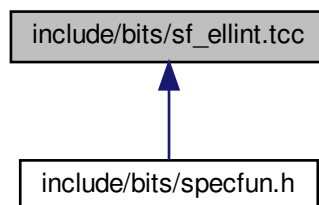
11.10 include/bits/sf_ellint.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- `#define _GLIBCXX_BITS_SF_ELLINT_TCC 1`

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__comp_ellint_1 (_Tp __k)`
Return the complete elliptic integral of the first kind $K(k)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp std::__detail::__comp_ellint_2 (_Tp __k)`
Return the complete elliptic integral of the second kind $E(k)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp std::__detail::__comp_ellint_3 (_Tp __k, _Tp __nu)`
Return the complete elliptic integral of the third kind $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp std::__detail::__comp_ellint_d (_Tp __k)`
- `template<typename _Tp >`
`_Tp std::__detail::__comp_ellint_rf (_Tp __x, _Tp __y)`
- `template<typename _Tp >`
`_Tp std::__detail::__comp_ellint_rg (_Tp __x, _Tp __y)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_1 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_2 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the second kind $E(k, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)`
Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_d (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_el1 (_Tp __x, _Tp __k_c)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_rc (_Tp __x, _Tp __y)`

Return the Carlson elliptic function $R_C(x, y) = R_F(x, y, y)$ where $R_F(x, y, z)$ is the Carlson elliptic function of the first kind.

- `template<typename _Tp >`
`_Tp std::__detail::__ellint_rd (_Tp __x, _Tp __y, _Tp __z)`

Return the Carlson elliptic function of the second kind $R_D(x, y, z) = R_J(x, y, z, z)$ where $R_J(x, y, z, p)$ is the Carlson elliptic function of the third kind.

- `template<typename _Tp >`
`_Tp std::__detail::__ellint_rf (_Tp __x, _Tp __y, _Tp __z)`

Return the Carlson elliptic function $R_F(x, y, z)$ of the first kind.

- `template<typename _Tp >`
`_Tp std::__detail::__ellint_rg (_Tp __x, _Tp __y, _Tp __z)`

Return the symmetric Carlson elliptic function of the second kind $R_G(x, y, z)$.

- `template<typename _Tp >`
`_Tp std::__detail::__ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)`

Return the Carlson elliptic function $R_J(x, y, z, p)$ of the third kind.

- `template<typename _Tp >`
`_Tp std::__detail::__heuman_lambda (_Tp __k, _Tp __phi)`

- `template<typename _Tp >`
`_Tp std::__detail::__jacobi_zeta (_Tp __k, _Tp __phi)`

11.10.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.10.2 Macro Definition Documentation

11.10.2.1 _GLIBCXX_BITS_SF_ELLINT_TCC

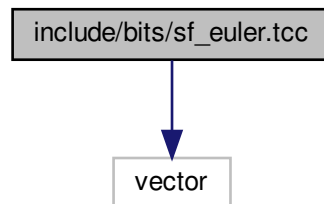
```
#define _GLIBCXX_BITS_SF_ELLINT_TCC 1
```

Definition at line 47 of file sf_ellint.tcc.

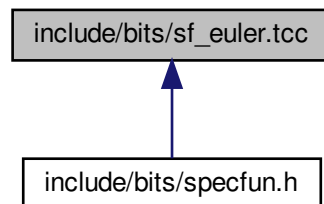
11.11 include/bits/sf_euler.tcc File Reference

```
#include <vector>
```

Include dependency graph for sf_euler.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_EULER_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__euler` (unsigned int __n)
This returns Euler number E_n .
- `template<typename _Tp >`
`_Tp std::__detail::__euler` (unsigned int __n, _Tp __x)
- `template<typename _Tp >`
`_Tp std::__detail::__euler_series` (unsigned int __n)
- `template<typename _Tp >`
`_Tp std::__detail::__eulerian_1` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__eulerian_1_recur` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__eulerian_2` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__eulerian_2_recur` (unsigned int __n, unsigned int __m)

11.11.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.11.2 Macro Definition Documentation

11.11.2.1 _GLIBCXX_BITS_SF_EULER_TCC

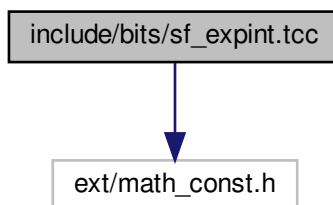
```
#define _GLIBCXX_BITS_SF_EULER_TCC 1
```

Definition at line 35 of file `sf_euler.tcc`.

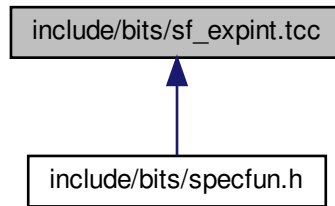
11.12 include/bits/sf_expint.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_EXPINT_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__coshint \(const _Tp __x\)](#)
Return the hyperbolic cosine integral $Chi(x)$.
- [template<typename _Tp > _Tp std::__detail::__expint \(unsigned int __n, _Tp __x\)](#)
Return the exponential integral $E_n(x)$.
- [template<typename _Tp > _Tp std::__detail::__expint \(_Tp __x\)](#)
Return the exponential integral $Ei(x)$.
- [template<typename _Tp > _Tp std::__detail::__expint_E1 \(_Tp __x\)](#)
Return the exponential integral $E_1(x)$.
- [template<typename _Tp > _Tp std::__detail::__expint_E1_asymp \(_Tp __x\)](#)
Return the exponential integral $E_1(x)$ by asymptotic expansion.
- [template<typename _Tp > _Tp std::__detail::__expint_E1_series \(_Tp __x\)](#)
Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.

- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei (_Tp __x)`
Return the exponential integral $Ei(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei_asymp (_Tp __x)`
Return the exponential integral $Ei(x)$ by asymptotic expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei_series (_Tp __x)`
Return the exponential integral $Ei(x)$ by series summation.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_asymp (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ for large argument.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_cont_frac (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ by continued fractions.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_large_n (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ for large order.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_recursion (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_series (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ by series summation.
- `template<typename _Tp >`
`_Tp std::__detail::__logint (const _Tp __x)`
Return the logarithmic integral $li(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__sinhint (const _Tp __x)`
Return the hyperbolic sine integral $Shi(x)$.

11.12.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.12.2 Macro Definition Documentation

11.12.2.1 _GLIBCXX_BITS_SF_EXPINT_TCC

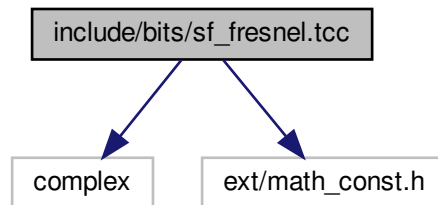
```
#define _GLIBCXX_BITS_SF_EXPINT_TCC 1
```

Definition at line 47 of file `sf_expint.tcc`.

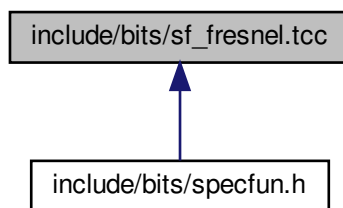
11.13 include/bits/sf_fresnel.tcc File Reference

```
#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](#)

Implementation-space details.

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.

11.13.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.13.2 Macro Definition Documentation

11.13.2.1 _GLIBCXX_BITS_SF_FRESNEL_TCC

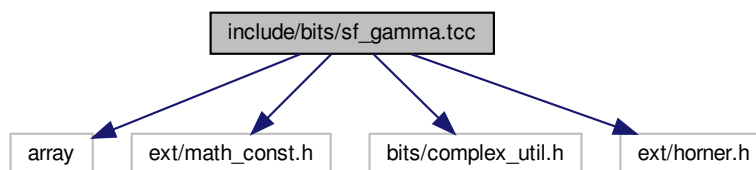
```
#define _GLIBCXX_BITS_SF_FRESNEL_TCC 1
```

Definition at line 31 of file `sf_fresnel.tcc`.

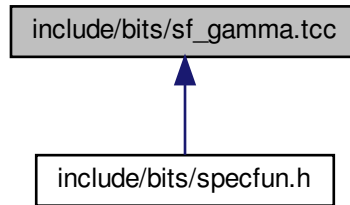
11.14 include/bits/sf_gamma.tcc File Reference

```
#include <array>
#include <ext/math_const.h>
#include <bits/complex_util.h>
#include <ext/horner.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](#)

Implementation-space details.

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 >`
`_Tp std::__detail::__digamma (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::__digamma (_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::__digamma_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::__digamma_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 >`
`_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_p (_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::__gamma_q (_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::__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 >`
`_Tp std::__detail::__hurwitz_zeta (_Tp __s, _Tp __a)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__lanczos_binet1p (_Tp __z)`

Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.

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)`
- `template<typename _Tp >`
`_Tp std::__detail::__log_binomial (unsigned int __n, unsigned int __k)`

Return the logarithm of the gamma function $\log(\Gamma(1 + z))$ by the Lanczos method.

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 __nu)`
- `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

$$a^{\underline{n}} = \Gamma(a + 1) / \Gamma(a - \nu + 1) = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1$$

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 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::__polygamma (unsigned int __m, _Tp __x)`

Return the polygamma function $\psi^{(m)}(x)$.

- `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

$$a^{\overline{n}} = \Gamma(a + n) / \Gamma(a) = \prod_{k=0}^{n-1} (a + k), (a)_0 = 1$$

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

$$a^{\overline{\nu}} = \Gamma(a + \nu) / \Gamma(a)$$

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::__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::__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`

11.14.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.14.2 Macro Definition Documentation

11.14.2.1 _GLIBCXX_BITS_SF_GAMMA_TCC

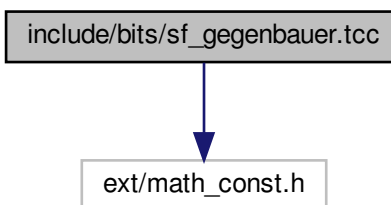
```
#define _GLIBCXX_BITS_SF_GAMMA_TCC 1
```

Definition at line 49 of file `sf_gamma.tcc`.

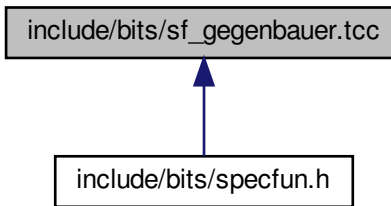
11.15 include/bits/sf_gegenbauer.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_GEGENBAUER_TCC 1](#)

Functions

- [template<typename _Tp > __gnu_cxx::__gegenbauer_t< _Tp > std::__detail::__gegenbauer_recur](#) (unsigned int __n, _Tp __lambda, _Tp __x)
- [template<typename _Tp > std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__gegenbauer_zeros](#) (unsigned int __n, _Tp __lambda)

11.15.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.15.2 Macro Definition Documentation

11.15.2.1 _GLIBCXX_BITS_SF_GEGENBAUER_TCC

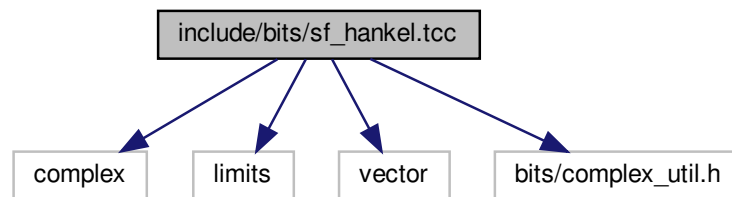
```
#define _GLIBCXX_BITS_SF_GEGENBAUER_TCC 1
```

Definition at line 31 of file sf_gegenbauer.tcc.

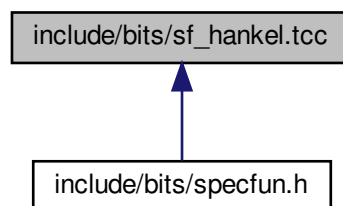
11.16 include/bits/sf_hankel.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- `#define _GLIBCXX_BITS_SF_HANKEL_TCC 1`

Functions

- `template<typename _Tp >`
`void std::__detail::__airy_arg (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)`
Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Bessel function.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_1 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Hankel function of the first kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_2 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_neumann (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Neumann function.
- `template<typename _Tp >`
`void std::__detail::__debye_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)`
- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn)`
- `template<typename _Tp >`
`void std::__detail::__hankel_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp > &__nup2, std::complex< _Tp > &__num2, std::complex< _Tp > &__num1d3, std::complex< _Tp > &__num2d3, std::complex< _Tp > &__num4d3, std::complex< _Tp > &__zeta, std::complex< _Tp > &__zetaphf, std::complex< _Tp > &__zetamhf, std::complex< _Tp > &__zetam3hf, std::complex< _Tp > &__zetrat)`
Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.
- `template<typename _Tp >`
`__gnu_cxx::__cyl_hankel_t< std::complex< _Tp >, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z)`

Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order `nu` along with their derivatives.

- `template<typename _Tp >`
`void std::__detail::__hankel_uniform_outer (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __↵`
`eps, std::complex< _Tp > &__zhat, std::complex< _Tp > &__1dnsq, std::complex< _Tp > &__num1d3, std↵`
`::complex< _Tp > &__num2d3, std::complex< _Tp > &__p, std::complex< _Tp > &__p2, std::complex< _Tp >`
`&__etm3h, std::complex< _Tp > &__etrat, std::complex< _Tp > &__Aip, std::complex< _Tp > &__o4dp, std↵`
`::complex< _Tp > &__Aim, std::complex< _Tp > &__o4dm, std::complex< _Tp > &__od2p, std::complex< _Tp`
`> &__od0dp, std::complex< _Tp > &__od2m, std::complex< _Tp > &__od0dm)`

Compute outer factors and associated functions of `z` and `nu` appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of `z` and `nu` returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.

- `template<typename _Tp >`
`void std::__detail::__hankel_uniform_sum (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex<`
`_Tp > __num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > __Aip, std::complex< _Tp > __o4dp,`
`std::complex< _Tp > __Aim, std::complex< _Tp > __o4dm, std::complex< _Tp > __od2p, std::complex< _Tp`
`> __od0dp, std::complex< _Tp > __od2m, std::complex< _Tp > __od0dm, _Tp __eps, std::complex< _Tp >`
`&__H1sum, std::complex< _Tp > &__H1psum, std::complex< _Tp > &__H2sum, std::complex< _Tp > &__H2psum)`

Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to `nterms` (less than 5) to achieve relative error `eps`.

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_bessel (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Bessel function.
- `template<typename _Tp >`
`__gnu_cxx::__sph_hankel_t< unsigned int, std::complex< _Tp >, std::complex< _Tp > > std::__detail::__↵`
`sph_hankel (unsigned int __n, std::complex< _Tp > __z)`
Helper to compute complex spherical Hankel functions and their derivatives.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Hankel function of the first kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_neumann (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Neumann function.

11.16.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.16.2 Macro Definition Documentation

11.16.2.1 _GLIBCXX_BITS_SF_HANKEL_TCC

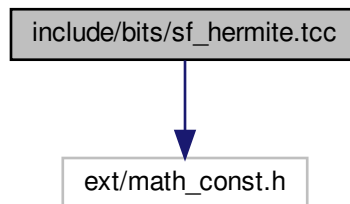
```
#define _GLIBCXX_BITS_SF_HANKEL_TCC 1
```

Definition at line 31 of file `sf_hankel.tcc`.

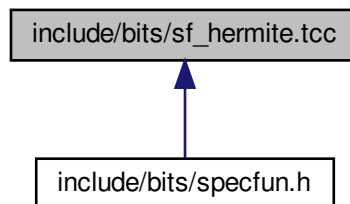
11.17 include/bits/sf_hermite.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_HERMITE_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__hermite` (unsigned int __n, _Tp __x)
This routine returns the Hermite polynomial of order n : $H_n(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__hermite_asymp` (unsigned int __n, _Tp __x)
This routine returns the Hermite polynomial of large order n : $H_n(x)$. We assume here that $x \geq 0$.
- `template<typename _Tp >`
`__gnu_cxx::__hermite_t<_Tp> std::__detail::__hermite_recur` (unsigned int __n, _Tp __x)
This routine returns the Hermite polynomial of order n : $H_n(x)$ by recursion on n .
- `template<typename _Tp >`
`std::vector< __gnu_cxx::__quadrature_point_t<_Tp> > std::__detail::__hermite_zeros` (unsigned int __n, _Tp __proto=_Tp{})
- `template<typename _Tp >`
`__gnu_cxx::__hermite_he_t<_Tp> std::__detail::__prob_hermite_recur` (unsigned int __n, _Tp __x)
This routine returns the Probabilists Hermite polynomial of order n : $He_n(x)$ by recursion on n .

11.17.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.17.2 Macro Definition Documentation

11.17.2.1 _GLIBCXX_BITS_SF_HERMITE_TCC

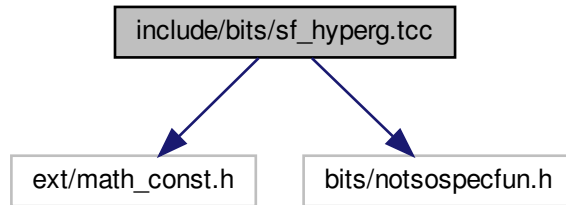
```
#define _GLIBCXX_BITS_SF_HERMITE_TCC 1
```

Definition at line 42 of file `sf_hermite.tcc`.

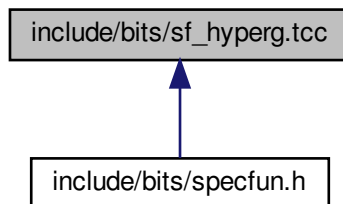
11.18 include/bits/sf_hyperg.tcc File Reference

```
#include <ext/math_const.h>
#include <bits/notsospecfun.h>
```

Include dependency graph for `sf_hyperrg.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_HYPERRG_TCC](#) 1

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__conf_hyperg (_Tp __a, _Tp __c, _Tp __x)`
Return the confluent hypergeometric function ${}_1F_1(a; c; x) = M(a, c, x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__conf_hyperg_lim (_Tp __c, _Tp __x)`
Return the confluent hypergeometric limit function ${}_0F_1(-; c; x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__conf_hyperg_lim_series (_Tp __c, _Tp __x)`
This routine returns the confluent hypergeometric limit function by series expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)`
Return the hypergeometric function ${}_1F_1(a; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.
- `template<typename _Tp >`
`_Tp std::__detail::__conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)`
This routine returns the confluent hypergeometric function by series expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`
Return the hypergeometric function ${}_2F_1(a, b; c; x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)`
Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.
- `template<typename _Tp >`
`_Tp std::__detail::__hyperg_recur (int __m, _Tp __b, _Tp __c, _Tp __x)`
Return the hypergeometric polynomial ${}_2F_1(-m, b; c; x)$ by Holm recursion.
- `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 `_GLIBCXX_BITS_SF_HYPERG_TCC`

```
#define _GLIBCXX_BITS_SF_HYPERG_TCC 1
```

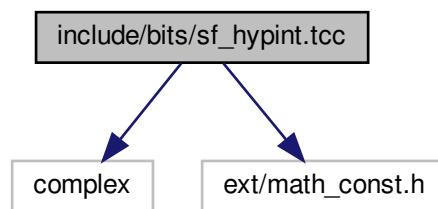
Definition at line 44 of file `sf_hyperg.tcc`.

11.19 `include/bits/sf_hypint.tcc` File Reference

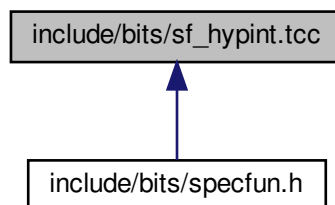
```
#include <complex>
```

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_HYPINT_TCC 1](#)

Functions

- `template<typename _Tp >
std::pair< _Tp, _Tp > std::__detail::__chshint (_Tp __x, _Tp &_Chi, _Tp &_Shi)`
This function returns the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals as a pair.
- `template<typename _Tp >
void std::__detail::__chshint_cont_frac (_Tp __t, _Tp &_Chi, _Tp &_Shi)`
This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by continued fraction for positive argument.
- `template<typename _Tp >
void std::__detail::__chshint_series (_Tp __t, _Tp &_Chi, _Tp &_Shi)`
This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by series summation for positive argument.

11.19.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.19.2 Macro Definition Documentation

11.19.2.1 _GLIBCXX_BITS_SF_HYPINT_TCC

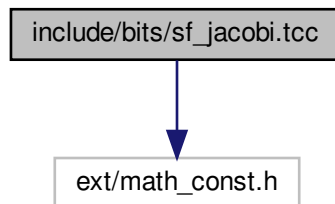
```
#define _GLIBCXX_BITS_SF_HYPINT_TCC 1
```

Definition at line 31 of file `sf_hypint.tcc`.

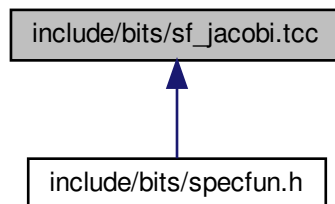
11.20 include/bits/sf_jacobi.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_JACOBI_TCC](#) 1

Functions

- `template<typename _Tp >`
`__gnu_cxx::__jacobi_t< _Tp > std::__detail::__jacobi_recur` (unsigned int __n, _Tp __alpha1, _Tp __beta1, _Tp __x)
- `template<typename _Tp >`
`std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__jacobi_zeros` (unsigned int __n, _Tp __alpha1, _Tp __beta1)
- `template<typename _Tp >`
`_Tp std::__detail::__radial_jacobi` (unsigned int __n, unsigned int __m, _Tp __rho)
- `template<typename _Tp >`
`std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__radial_jacobi_zeros` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tp > std::__detail::__zernike` (unsigned int __n, int __m, _Tp __rho, _Tp __phi)

11.20.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.20.2 Macro Definition Documentation

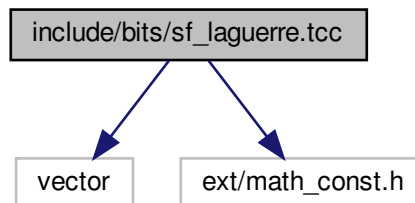
11.20.2.1 _GLIBCXX_BITS_SF_JACOBI_TCC

```
#define _GLIBCXX_BITS_SF_JACOBI_TCC 1
```

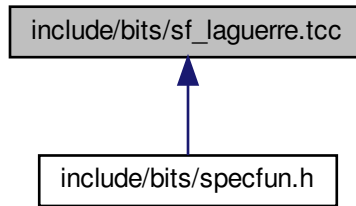
Definition at line 31 of file `sf_jacobi.tcc`.

11.21 include/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](#)

Implementation-space details.

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 degree n , order m : $L_n^{(m)}(x)$.
- `template<typename _Tpa, typename _Tp >`
`_Tp std::__detail::__laguerre (unsigned int __n, _Tpa __alpha1, _Tp __x)`
This routine returns the associated Laguerre polynomial of degree n , order α : $L_n^{(\alpha)}(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__laguerre (unsigned int __n, _Tp __x)`
This routine returns the Laguerre polynomial of degree n : $L_n(x)$.
- `template<typename _Tpa, typename _Tp >`
`_Tp std::__detail::__laguerre_hyperg (unsigned int __n, _Tpa __alpha1, _Tp __x)`
Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.
- `template<typename _Tpa, typename _Tp >`
`_Tp std::__detail::__laguerre_large_n (unsigned __n, _Tpa __alpha1, _Tp __x)`
This routine returns the associated Laguerre polynomial of degree n , order $\alpha > -1$ for large n . Abramowitz & Stegun, 13.5.21.
- `template<typename _Tpa, typename _Tp >`
`__gnu_cxx::__laguerre_t< _Tpa, _Tp > std::__detail::__laguerre_recur (unsigned int __n, _Tpa __alpha1, _Tp __x)`
This routine returns the associated Laguerre polynomial of degree n , order α : $L_n^{(\alpha)}(x)$ by recursion.
- `template<typename _Tp >`
`std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__laguerre_zeros (unsigned int __n, _Tp __alpha1)`

11.21.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.21.2 Macro Definition Documentation

11.21.2.1 _GLIBCXX_BITS_SF_LAGUERRE_TCC

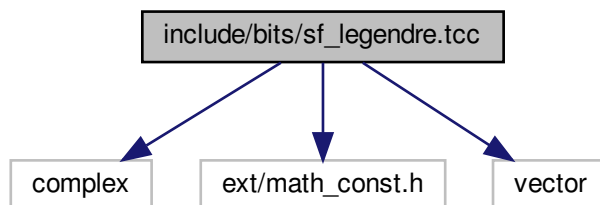
```
#define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1
```

Definition at line 44 of file `sf_laguerre.tcc`.

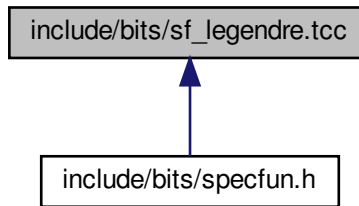
11.22 include/bits/sf_legendre.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__assoc_legendre_p \(unsigned int __l, unsigned int __m, _Tp __x, _Tp __phase=_Tp{+1}\)](#)
Return the associated Legendre function by recursion on l and downward recursion on m .
- [template<typename _Tp > __gnu_cxx::__legendre_p_t< _Tp > std::__detail::__legendre_p \(unsigned int __l, _Tp __x\)](#)
Return the Legendre polynomial by upward recursion on degree l .
- [template<typename _Tp > _Tp std::__detail::__legendre_q \(unsigned int __l, _Tp __x\)](#)
Return the Legendre function of the second kind by upward recursion on degree l .
- [template<typename _Tp > std::vector< __gnu_cxx::__quadrature_point_t< _Tp > > std::__detail::__legendre_zeros \(unsigned int __l, _Tp proto=_Tp{}\)](#)
- [template<typename _Tp > std::complex< _Tp > std::__detail::__sph_harmonic \(unsigned int __l, int __m, _Tp __theta, _Tp __phi\)](#)
Return the spherical harmonic function.
- [template<typename _Tp > _Tp std::__detail::__sph_legendre \(unsigned int __l, unsigned int __m, _Tp __theta\)](#)
Return the spherical associated Legendre function.

11.22.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.22.2 Macro Definition Documentation

11.22.2.1 _GLIBCXX_BITS_SF_LEGENDRE_TCC

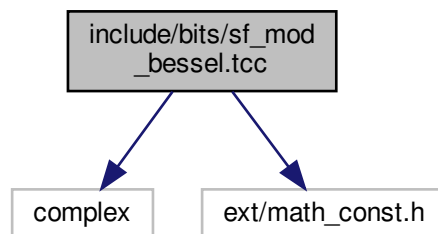
```
#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1
```

Definition at line 47 of file `sf_legendre.tcc`.

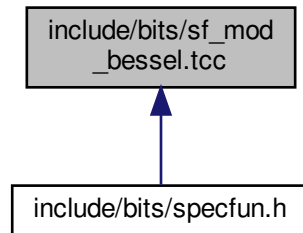
11.23 include/bits/sf_mod_bessel.tcc File Reference

```
#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`

Implementation-space details.

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_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 std::__detail::__cyl_bessel_k(_Tp __nu, _Tp __x)`
Return the irregular modified Bessel function $K_\nu(x)$ of order ν .
- `template<typename _Tp >`
`__gnu_cxx::__fock_airy_t<_Tp, std::complex<_Tp > > std::__detail::__fock_airy(_Tp __x)`
Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w_1'(x)$ and $w_2'(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$
- `template<typename _Tp >`
`__gnu_cxx::__sph_mod_bessel_t< unsigned int, _Tp, _Tp > std::__detail::__sph_bessel_ik(unsigned int __n, _Tp __x)`
Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i_n'(x)$ and $k_n'(x)$ respectively.

11.23.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.23.2 Macro Definition Documentation

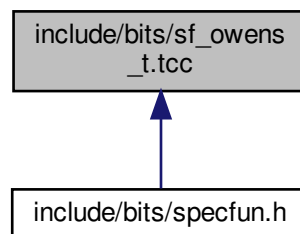
11.23.2.1 _GLIBCXX_BITS_SF_MOD_BESSEL_TCC

```
#define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1
```

Definition at line 47 of file `sf_mod_bessel.tcc`.

11.24 include/bits/sf_owens_t.tcc File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_OWENS_T_TCC 1](#)

Functions

- [template<typename _Tp >
_Tp std::__detail::__gauss \(_Tp __x\)](#)
- [template<typename _Tp >
_Tp std::__detail::__owens_t \(_Tp __h, _Tp __a\)](#)
- [template<typename _Tp >
_Tp std::__detail::__znorm1 \(_Tp __x\)](#)
- [template<typename _Tp >
_Tp std::__detail::__znorm2 \(_Tp __x\)](#)

11.24.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.24.2 Macro Definition Documentation

11.24.2.1 [_GLIBCXX_BITS_SF_OWENS_T_TCC](#)

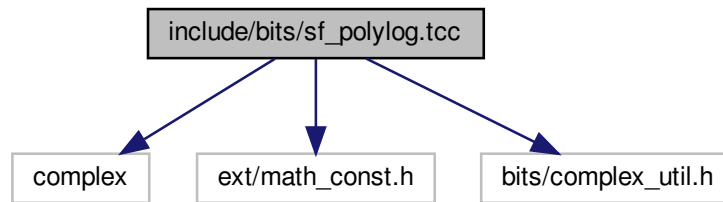
```
#define _GLIBCXX_BITS_SF_OWENS_T_TCC 1
```

Definition at line 31 of file `sf_owens_t.tcc`.

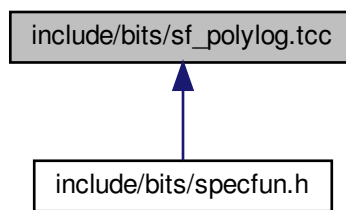
11.25 include/bits/sf_polylog.tcc File Reference

```
#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`

Implementation-space details.

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::fp_promote_t< std::complex< _Tp >, _ArgType > std::__detail::__polylog_exp (_Tp __s, _ArgType __w)`
- `template<typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >
std::complex< _Tp > std::__detail::__polylog_exp_neg (int __n, std::complex< _Tp > __w)`

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_int (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_int (int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_real (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_real (_Tp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos_int (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos_int (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos_real (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos_real (_Tp __s, _Tp __w)`
- `template<typename _PowTp, typename _Tp >`
`_Tp std::__detail::__polylog_exp_sum (_PowTp __s, _Tp __w)`

11.25.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.25.2 Macro Definition Documentation

11.25.2.1 _GLIBCXX_BITS_SF_POLYLOG_TCC

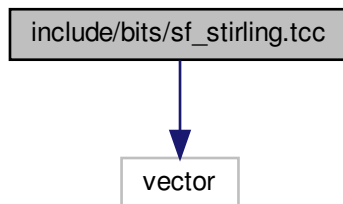
```
#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1
```

Definition at line 41 of file `sf_polylog.tcc`.

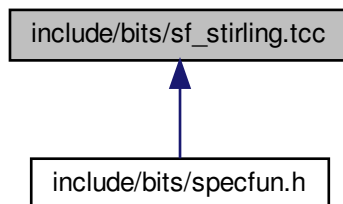
11.26 include/bits/sf_stirling.tcc File Reference

```
#include <vector>
```

Include dependency graph for sf_stirling.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_STIRLING_TCC](#) 1

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__log_stirling_1` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__log_stirling_1_sign` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__log_stirling_2` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__stirling_1` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__stirling_1_recur` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__stirling_1_series` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__stirling_2` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__stirling_2_recur` (unsigned int __n, unsigned int __m)
- `template<typename _Tp >`
`_Tp std::__detail::__stirling_2_series` (unsigned int __n, unsigned int __m)

11.26.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.26.2 Macro Definition Documentation

11.26.2.1 _GLIBCXX_BITS_SF_STIRLING_TCC

```
#define _GLIBCXX_BITS_SF_STIRLING_TCC 1
```

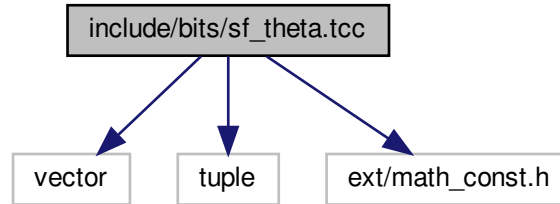
Definition at line 35 of file `sf_stirling.tcc`.

11.27 include/bits/sf_theta.tcc File Reference

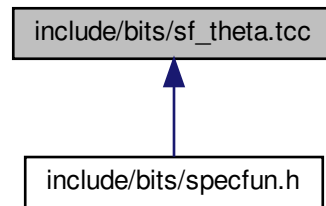
```
#include <vector>
#include <tuple>
```

```
#include <ext/math_const.h>
```

Include dependency graph for sf_theta.tcc:



This graph shows which files directly or indirectly include this file:



Classes

- struct [std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >](#)
- struct [std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__arg_t](#)
- struct [std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau_t](#)
- struct [std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >](#)
- struct [std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >](#)
- struct [std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >](#)

Namespaces

- [std](#)
- [std::__detail](#)

Implementation-space details.

Macros

- `#define _GLIBCXX_BITS_SF_THETA_TCC 1`

Functions

- `template<typename _Tp >
_Tp std::__detail::__ellnome (_Tp __k)`
- `template<typename _Tp >
_Tp std::__detail::__ellnome_k (_Tp __k)`
- `template<typename _Tp >
_Tp std::__detail::__ellnome_series (_Tp __k)`
- `template<typename _Tp >
__gnu_cxx::__jacobi_ellint_t< _Tp > std::__detail::__jacobi_ellint (_Tp __k, _Tp __u)`
- `template<typename _Tp >
std::complex< _Tp > std::__detail::__jacobi_theta_1 (std::complex< _Tp > __q, std::complex< _Tp > __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_1 (_Tp __q, const _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_1_prod (_Tp __q, _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_1_sum (_Tp __q, _Tp __x)`
- `template<typename _Tp >
std::complex< _Tp > std::__detail::__jacobi_theta_2 (std::complex< _Tp > __q, std::complex< _Tp > __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_2 (_Tp __q, const _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_2_prod (_Tp __q, _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_2_sum (_Tp __q, _Tp __x)`
- `template<typename _Tp >
std::complex< _Tp > std::__detail::__jacobi_theta_3 (std::complex< _Tp > __q, std::complex< _Tp > __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_3 (_Tp __q, const _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_3_prod (_Tp __q, _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_3_sum (_Tp __q, _Tp __x)`
- `template<typename _Tp >
std::complex< _Tp > std::__detail::__jacobi_theta_4 (std::complex< _Tp > __q, std::complex< _Tp > __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_4 (_Tp __q, const _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_4_prod (_Tp __q, _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__jacobi_theta_4_sum (_Tp __q, _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__theta_1 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__theta_2 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >
_Tp std::__detail::__theta_2_asymp (_Tp __nu, _Tp __x)`

- `template<typename _Tp >`
`_Tp std::__detail::__theta_2_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_3 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_3_asymp (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_3_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_4 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_c (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_d (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_n (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_s (_Tp __k, _Tp __x)`

11.27.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.27.2 Macro Definition Documentation

11.27.2.1 `_GLIBCXX_BITS_SF_THETA_TCC`

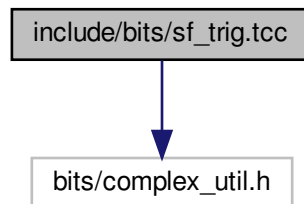
```
#define _GLIBCXX_BITS_SF_THETA_TCC 1
```

Definition at line 31 of file `sf_theta.tcc`.

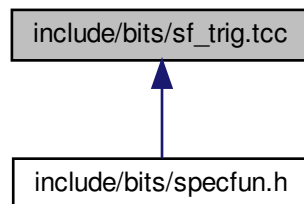
11.28 include/bits/sf_trig.tcc File Reference

```
#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](#)

Implementation-space details.

Macros

- [#define _GLIBCXX_BITS_SF_TRIG_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__cos_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cos_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp std::__detail::__cosh_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cosh_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polar_pi (_Tp __rho, _Tp __phi_pi)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polar_pi (_Tp __rho, const std::complex< _Tp > &__phi_pi)`
- `template<typename _Tp >`
`_Tp std::__detail::__sin_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sin_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< _Tp > std::__detail::__sincos (_Tp __x)`
- `template<>`
`__gnu_cxx::__sincos_t< float > std::__detail::__sincos (float __x)`
- `template<>`
`__gnu_cxx::__sincos_t< double > std::__detail::__sincos (double __x)`
- `template<>`
`__gnu_cxx::__sincos_t< long double > std::__detail::__sincos (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< _Tp > std::__detail::__sincos_pi (_Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__sinh_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sinh_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp std::__detail::__tan_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__tan_pi (std::complex< _Tp > __z)`
- `template<typename _Tp >`
`_Tp std::__detail::__tanh_pi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__tanh_pi (std::complex< _Tp > __z)`

11.28.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

11.28.2 Macro Definition Documentation

11.28.2.1 _GLIBCXX_BITS_SF_TRIG_TCC

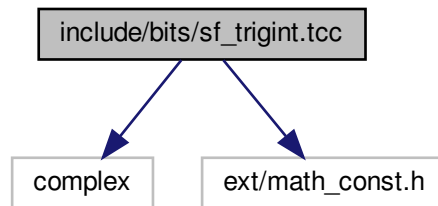
```
#define _GLIBCXX_BITS_SF_TRIG_TCC 1
```

Definition at line 31 of file sf_trig.tcc.

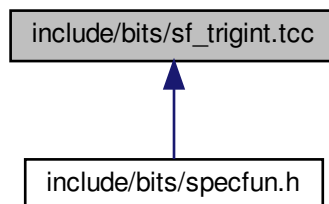
11.29 include/bits/sf_trigint.tcc File Reference

```
#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`

Implementation-space details.

Macros

- `#define _GLIBCXX_BITS_SF_TRIGINT_TCC 1`

Functions

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > std::__detail::__sincosint (_Tp __x)`
This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a pair.
- `template<typename _Tp >`
`void std::__detail::__sincosint_asymp (_Tp __t, _Tp &_Si, _Tp &_Ci)`
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.
- `template<typename _Tp >`
`void std::__detail::__sincosint_cont_frac (_Tp __t, _Tp &_Si, _Tp &_Ci)`
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.
- `template<typename _Tp >`
`void std::__detail::__sincosint_series (_Tp __t, _Tp &_Si, _Tp &_Ci)`
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.

11.29.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.29.2 Macro Definition Documentation

11.29.2.1 _GLIBCXX_BITS_SF_TRIGINT_TCC

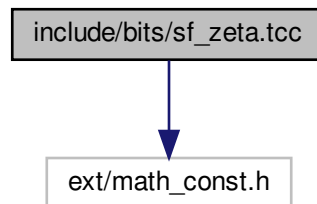
```
#define _GLIBCXX_BITS_SF_TRIGINT_TCC 1
```

Definition at line 31 of file `sf_trigint.tcc`.

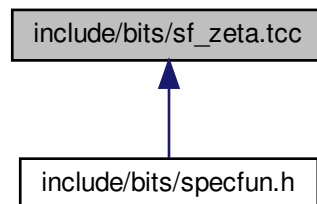
11.30 include/bits/sf_zeta.tcc File Reference

```
#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](#)

Implementation-space details.

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::__exp2 (_Tp __x)`
- `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_laurent (_Tp __s)`
Compute the Riemann zeta function $\zeta(s)$ by Laurent expansion about $s = 1$.
- `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 size_t std::__detail::__Num_Stieltjes = 21`
- `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_Stieltjes [_Num_Stieltjes]`
- `constexpr long double std::__detail::__S_zetam1 [_S_num_zetam1]`

11.30.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.30.2 Macro Definition Documentation

11.30.2.1 _GLIBCXX_BITS_SF_ZETA_TCC

```
#define _GLIBCXX_BITS_SF_ZETA_TCC 1
```

Definition at line 46 of file `sf_zeta.tcc`.

11.31 include/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_prime.tcc>
#include <bits/sf_trig.tcc>
#include <bits/sf_bernoulli.tcc>
#include <bits/sf_gamma.tcc>
#include <bits/sf_euler.tcc>
#include <bits/sf_stirling.tcc>
#include <bits/sf_bessel.tcc>
#include <bits/sf_beta.tcc>
#include <bits/sf_cardinal.tcc>
#include <bits/sf_chebyshev.tcc>
#include <bits/sf_coulomb.tcc>
#include <bits/sf_dawson.tcc>
#include <bits/sf_ellint.tcc>
#include <bits/sf_expint.tcc>
#include <bits/sf_fresnel.tcc>
#include <bits/sf_gegenbauer.tcc>
#include <bits/sf_hyperg.tcc>
#include <bits/sf_hypint.tcc>
#include <bits/sf_jacobi.tcc>
#include <bits/sf_laguerre.tcc>
#include <bits/sf_legendre.tcc>
#include <bits/sf_lerch.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::fp_promote_t< _Tp > __gnu_cxx::airy_ai (_Tp __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::airy_bi (_Tp __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::binomial_p (_Tp __p, unsigned int __n, unsigned int __k)`
Return the binomial cumulative distribution function.
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::clausen (unsigned int __m, _Tp __x)`
- `template<typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_t< _Tp > > __gnu_cxx::clausen (unsigned int __m, std::complex< _Tp > __z)`

- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rg (_Tx __x, _Ty __y)`
- `template<typename _Tpa, typename _Tpc, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tpa, _Tpc, _Tp > __gnu_cxx::conf_hyperg (_Tpa __a, _Tpc __c, _Tp __x)`
- `template<typename _Tpc, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tpc, _Tp > __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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::digamma` (_Tp __x)
- `float __gnu_cxx::digammaf` (float __x)
- `long double __gnu_cxx::digammal` (long double __x)
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > std::expint (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_t< _Tlam, _Tp > __gnu_cxx::exponential_p (_Tlam __lambda, _Tp __x)`
Return the exponential cumulative probability density function.
- `template<typename _Tlam, typename _Tp >`
`__gnu_cxx::fp_promote_t< _Tlam, _Tp > __gnu_cxx::exponential_pdf (_Tlam __lambda, _Tp __x)`
Return the exponential probability density function.
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_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

$$a^{\overline{n}} = \prod_{k=0}^{n-1} (a - k), a^{\overline{0}} = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $n^{\overline{n}} = n!$.

- `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::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::fisher_f_p (_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::fp_promote_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::fp_promote_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::fp_promote_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 _Tp >
[__gnu_cxx::fp_promote_t<_Ta, _Tp> __gnu_cxx::gamma_p](#) (_Ta __a, _Tp __x)
Return the gamma cumulative propability distribution function or the regularized lower incomplete gamma function.
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::fp_promote_t<_Ta, _Tb, _Tp> __gnu_cxx::gamma_pdf](#) (_Ta __alpha, _Tb __beta, _Tp __x)
Return the gamma propability distribution function.
- float [__gnu_cxx::gamma_pf](#) (float __a, float __x)
- long double [__gnu_cxx::gamma_pl](#) (long double __a, long double __x)
- template<typename _Ta, typename _Tp >
[__gnu_cxx::fp_promote_t<_Ta, _Tp> __gnu_cxx::gamma_q](#) (_Ta __a, _Tp __x)
Return the gamma complementary cumulative propability distribution (or survival) function or the regularized upper incomplete gamma function.
- float [__gnu_cxx::gamma_qf](#) (float __a, float __x)
- long double [__gnu_cxx::gamma_ql](#) (long double __a, long double __x)
- template<typename _Ta >
[__gnu_cxx::fp_promote_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 _Tlam, typename _Tp >
[__gnu_cxx::fp_promote_t<_Tlam, _Tp> __gnu_cxx::gegenbauer](#) (unsigned int __n, _Tlam __lambda, _Tp __x)
- float [__gnu_cxx::gegenbauerf](#) (unsigned int __n, float __lambda, float __x)
- long double [__gnu_cxx::gegenbauerl](#) (unsigned int __n, long double __lambda, long double __x)
- template<typename _Tp >
[__gnu_cxx::fp_promote_t<_Tp> __gnu_cxx::harmonic](#) (unsigned int __n)
- template<typename _Tp >
[__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t<_Ta, _Tb, _Tp> __gnu_cxx::ibeta](#) (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::fp_promote_t<_Ta, _Tb, _Tp> __gnu_cxx::ibetac](#) (_Ta __a, _Tb __b, _Tp __x)
- float [__gnu_cxx::ibetaf](#) (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::fp_promote_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi](#) (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)
- template<typename _Kp , typename _Up >
[__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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 _Tpq , typename _Tp >
[__gnu_cxx::fp_promote_t< _Tpq, _Tp > __gnu_cxx::jacobi_theta_1](#) (_Tpq __q, _Tp __x)
- float [__gnu_cxx::jacobi_theta_1f](#) (float __q, float __x)
- long double [__gnu_cxx::jacobi_theta_1l](#) (long double __q, long double __x)
- template<typename _Tpq , typename _Tp >
[__gnu_cxx::fp_promote_t< _Tpq, _Tp > __gnu_cxx::jacobi_theta_2](#) (_Tpq __q, _Tp __x)
- float [__gnu_cxx::jacobi_theta_2f](#) (float __q, float __x)
- long double [__gnu_cxx::jacobi_theta_2l](#) (long double __q, long double __x)
- template<typename _Tpq , typename _Tp >
[__gnu_cxx::fp_promote_t< _Tpq, _Tp > __gnu_cxx::jacobi_theta_3](#) (_Tpq __q, _Tp __x)
- float [__gnu_cxx::jacobi_theta_3f](#) (float __q, float __x)
- long double [__gnu_cxx::jacobi_theta_3l](#) (long double __q, long double __x)
- template<typename _Tpq , typename _Tp >
[__gnu_cxx::fp_promote_t< _Tpq, _Tp > __gnu_cxx::jacobi_theta_4](#) (_Tpq __q, _Tp __x)
- float [__gnu_cxx::jacobi_theta_4f](#) (float __q, float __x)
- long double [__gnu_cxx::jacobi_theta_4l](#) (long double __q, long double __x)
- template<typename _Tk , typename _Tphi >
[__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t<_Tp > std::legendre](#) (unsigned int __l, _Tp __x)
- template<typename _Tp >
[__gnu_cxx::fp_promote_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, typename _Ts, typename _Ta >
[__gnu_cxx::fp_promote_t<_Tp, _Ts, _Ta > __gnu_cxx::lerch_phi](#) (_Tp __z, _Ts __s, _Ta __a)
- float [__gnu_cxx::lerch_phif](#) (float __z, float __s, float __a)
- long double [__gnu_cxx::lerch_phil](#) (long double __z, long double __s, long double __a)
- template<typename _Tp >
[__gnu_cxx::fp_promote_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::fp_promote_t<_Tp, _Tnu > __gnu_cxx::lfalling_factorial](#) (_Tp __a, _Tnu __nu)
Return the logarithm of the falling factorial function or the lower Pochhammer symbol. The falling factorial function is defined by

$$a^{\underline{n}} = \Gamma(a+1)/\Gamma(a-\nu+1) = \prod_{k=0}^{n-1} (a-k), a^{\underline{0}} = 1$$

In particular, $n^{\underline{n}} = n!$. Thus this function returns

$$\ln[a^{\underline{n}}] = \ln[\Gamma(a+1)] - \ln[\Gamma(a-\nu+1)], \ln[a^{\underline{0}}] = 0$$

Many notations exist for this function: $(a)_{\nu}$,

$$\left\{ \begin{matrix} a \\ \nu \end{matrix} \right\}$$

, 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::fp_promote_t<_Ta > __gnu_cxx::lgamma](#) (_Ta __a)
- template<typename _Ta >
[std::complex<__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t](#)< _Ta, _Tb, _Tp > [__gnu_cxx::logistic_p](#) (_Ta __a, _Tb __b, _Tp __x)
Return the logistic cumulative distribution function.
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::fp_promote_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::fp_promote_t](#)< _Tmu, _Tsig, _Tp > [__gnu_cxx::lognormal_p](#) (_Tmu __mu, _Tsig __sigma, _Tp __x)
Return the lognormal cumulative probability density function.
- template<typename _Tmu, typename _Tsig, typename _Tp >
[__gnu_cxx::fp_promote_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::fp_promote_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),

$$\begin{bmatrix} a \\ \nu \end{bmatrix}$$

, 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::fp_promote_t](#)< _Tmu, _Tsig, _Tp > [__gnu_cxx::normal_p](#) (_Tmu __mu, _Tsig __sigma, _Tp __x)
Return the normal cumulative probability density function.
- template<typename _Tmu, typename _Tsig, typename _Tp >
[__gnu_cxx::fp_promote_t](#)< _Tmu, _Tsig, _Tp > [__gnu_cxx::normal_pdf](#) (_Tmu __mu, _Tsig __sigma, _Tp __x)
Return the gamma cumulative probability distribution function.
- template<typename _Tph, typename _Tpa >
[__gnu_cxx::fp_promote_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 _Tp >
[__gnu_cxx::fp_promote_t](#)< _Tp > [__gnu_cxx::polygamma](#) (unsigned int __m, _Tp __x)
- float [__gnu_cxx::polygammaf](#) (unsigned int __m, float __x)
- long double [__gnu_cxx::polygammal](#) (unsigned int __m, long double __x)

- `template<typename _Tp, typename _Wp >`
`__gnu_cxx::fp_promote_t< _Tp, _Wp > __gnu_cxx::polylog (_Tp __s, _Wp __w)`
- `template<typename _Tp, typename _Wp >`
`std::complex< __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::sinc (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::fp_promote_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::fp_promote_t< _Tp > > __gnu_cxx::sincos (_Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__sincos_t< __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::sinhc](#) (_Tp __x)
- template<typename _Tp >
 [__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > std::sph_bessel](#) (unsigned int __n, _Tp __x)
- template<typename _Tp >
 [__gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > > __gnu_cxx::sph_hankel_1](#) (unsigned int __n, _Tp __z)
- template<typename _Tp >
 [std::complex< __gnu_cxx::fp_promote_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::fp_promote_t< _Tp > > __gnu_cxx::sph_hankel_2](#) (unsigned int __n, _Tp __z)
- template<typename _Tp >
 [std::complex< __gnu_cxx::fp_promote_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::fp_promote_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_harmonici` (unsigned int __l, int __m, long double __theta, long double __phi)
- `template<typename _Tp > __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_t< _Tp > __gnu_cxx::student_t_p` (_Tt __t, unsigned int __nu)
Return the Students T probability function.
- `template<typename _Tt, typename _Tp > __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Ta > __gnu_cxx::tgamma` (_Ta __a)
- `template<typename _Ta > std::complex< __gnu_cxx::fp_promote_t< _Ta > > __gnu_cxx::tgamma` (std::complex< _Ta > __a)
- `template<typename _Ta, typename _Tp > __gnu_cxx::fp_promote_t< _Ta, _Tp > __gnu_cxx::tgamma` (_Ta __a, _Tp __x)
- `template<typename _Ta, typename _Tp > __gnu_cxx::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_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::fp_promote_t< _Ta, _Tb, _Tp > __gnu_cxx::weibull_p](#) (_Ta __a, _Tb __b, _Tp __x)
Return the Weibull cumulative probability density function.
- template<typename _Ta, typename _Tb, typename _Tp >
[__gnu_cxx::fp_promote_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::fp_promote_t< _Trho, _Tphi > __gnu_cxx::zernike](#) (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)
- float [__gnu_cxx::zernikef](#) (unsigned int __n, int __m, float __rho, float __phi)
- long double [__gnu_cxx::zernikel](#) (unsigned int __n, int __m, long double __rho, long double __phi)

11.31.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

11.31.2 Macro Definition Documentation

11.31.2.1 __cpp_lib_math_special_functions

```
#define __cpp_lib_math_special_functions 201603L
```

Definition at line 39 of file specfun.h.

11.31.2.2 __STDCPP_MATH_SPEC_FUNCS__

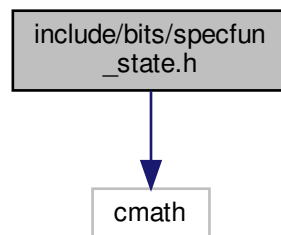
```
#define __STDCPP_MATH_SPEC_FUNCS__ 201003L
```

Definition at line 37 of file specfun.h.

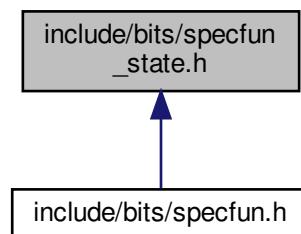
11.32 include/bits/specfun_state.h File Reference

```
#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::__chebyshev_t< _Tp >](#)
- struct [__gnu_cxx::__chebyshev_u_t< _Tp >](#)
- struct [__gnu_cxx::__chebyshev_v_t< _Tp >](#)
- struct [__gnu_cxx::__chebyshev_w_t< _Tp >](#)
- struct [__gnu_cxx::__cyl_bessel_t< _Tnu, _Tx, _Tp >](#)
- struct [__gnu_cxx::__cyl_coulomb_t< _Teta, _Trho, _Tp >](#)
- struct [__gnu_cxx::__cyl_hankel_t< _Tnu, _Tx, _Tp >](#)
- struct [__gnu_cxx::__cyl_mod_bessel_t< _Tnu, _Tx, _Tp >](#)
- struct [__gnu_cxx::__fock_airy_t< _Tx, _Tp >](#)
- struct [__gnu_cxx::__gamma_inc_t< _Tp >](#)
- struct [__gnu_cxx::__gamma_temme_t< _Tp >](#)

A structure for the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ 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::__gappa_pq_t< _Tp >](#)
- struct [__gnu_cxx::__gegenbauer_t< _Tp >](#)
- struct [__gnu_cxx::__hermite_he_t< _Tp >](#)
- struct [__gnu_cxx::__hermite_t< _Tp >](#)
- struct [__gnu_cxx::__jacobi_ellint_t< _Tp >](#)
- struct [__gnu_cxx::__jacobi_t< _Tp >](#)
- struct [__gnu_cxx::__laguerre_t< _Tpa, _Tp >](#)
- struct [__gnu_cxx::__legendre_p_t< _Tp >](#)
- struct [__gnu_cxx::__lgamma_t< _Tp >](#)
- struct [__gnu_cxx::__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](#)

Enumerations

- enum [__gnu_cxx::gauss_quad_type](#) { [__gnu_cxx::Gauss](#), [__gnu_cxx::Gauss_Lobatto](#), [__gnu_cxx::Gauss_Radau_lower](#), [__gnu_cxx::Gauss_Radau_upper](#) }

Enumeration for differing types of Gauss quadrature. The `gauss_quad_type` is used to determine the boundary condition modifications applied to orthogonal polynomials for quadrature rules.

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 include/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`, [535](#)
- `_Airy_asymp`
 - `std::__detail::_Airy_asymp`, [538](#)
- `_Airy_asymp_series`
 - `std::__detail::_Airy_asymp_series`, [545](#)
- `_AsympTerminator`
 - `std::__detail::_AsympTerminator`, [563](#)
- `_Cmplx`
 - `std::__detail::_jacobi_lattice_t`, [514](#)
 - `std::__detail::_jacobi_theta_0_t`, [523](#)
 - `std::__detail::_weierstrass_invariants_t`, [528](#)
 - `std::__detail::_weierstrass_roots_t`, [531](#)
 - `std::__detail::_Airy_asymp`, [538](#)
 - `std::__detail::_Airy_series`, [550](#)
- `_GLIBCXX_BITS_SF_AIRY_TCC`
 - `sf_airy.tcc`, [571](#)
- `_GLIBCXX_BITS_SF_BERNOULLI_TCC`
 - `sf_bernoulli.tcc`, [572](#)
- `_GLIBCXX_BITS_SF_BESSEL_TCC`
 - `sf_bessel.tcc`, [575](#)
- `_GLIBCXX_BITS_SF_BETA_TCC`
 - `sf_beta.tcc`, [576](#)
- `_GLIBCXX_BITS_SF_CARDINAL_TCC`
 - `sf_cardinal.tcc`, [578](#)
- `_GLIBCXX_BITS_SF_CHEBYSHEV_TCC`
 - `sf_chebyshev.tcc`, [580](#)
- `_GLIBCXX_BITS_SF_COULOMB_TCC`
 - `sf_coulomb.tcc`, [582](#)
- `_GLIBCXX_BITS_SF_DAWSON_TCC`
 - `sf_dawson.tcc`, [583](#)
- `_GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC`
 - `sf_distributions.tcc`, [587](#)
- `_GLIBCXX_BITS_SF_ELLINT_TCC`
 - `sf_ellint.tcc`, [589](#)
- `_GLIBCXX_BITS_SF_EULER_TCC`
 - `sf_euler.tcc`, [591](#)
- `_GLIBCXX_BITS_SF_EXPINT_TCC`
 - `sf_expint.tcc`, [593](#)
- `_GLIBCXX_BITS_SF_FRESNEL_TCC`
 - `sf_fresnel.tcc`, [595](#)
- `_GLIBCXX_BITS_SF_GAMMA_TCC`
 - `sf_gamma.tcc`, [603](#)
- `_GLIBCXX_BITS_SF_GEGENBAUER_TCC`
 - `sf_gegenbauer.tcc`, [604](#)
- `_GLIBCXX_BITS_SF_HANKEL_TCC`
 - `sf_hankel.tcc`, [607](#)
- `_GLIBCXX_BITS_SF_HERMITE_TCC`
 - `sf_hermite.tcc`, [609](#)
- `_GLIBCXX_BITS_SF_HYPERG_TCC`
 - `sf_hyperg.tcc`, [612](#)
- `_GLIBCXX_BITS_SF_HYPINT_TCC`
 - `sf_hypint.tcc`, [613](#)
- `_GLIBCXX_BITS_SF_JACOBI_TCC`
 - `sf_jacobi.tcc`, [615](#)
- `_GLIBCXX_BITS_SF_LAGUERRE_TCC`
 - `sf_laguerre.tcc`, [617](#)
- `_GLIBCXX_BITS_SF_LEGENDRE_TCC`
 - `sf_legendre.tcc`, [619](#)
- `_GLIBCXX_BITS_SF_MOD_BESSEL_TCC`
 - `sf_mod_bessel.tcc`, [621](#)
- `_GLIBCXX_BITS_SF_OWENS_T_TCC`
 - `sf_owens.t.tcc`, [622](#)
- `_GLIBCXX_BITS_SF_POLYLOG_TCC`
 - `sf_polylog.tcc`, [625](#)
- `_GLIBCXX_BITS_SF_STIRLING_TCC`
 - `sf_stirling.tcc`, [627](#)
- `_GLIBCXX_BITS_SF_THETA_TCC`
 - `sf_theta.tcc`, [630](#)
- `_GLIBCXX_BITS_SF_TRIGINT_TCC`
 - `sf_trigint.tcc`, [634](#)
- `_GLIBCXX_BITS_SF_TRIG_TCC`
 - `sf_trig.tcc`, [632](#)
- `_GLIBCXX_BITS_SF_ZETA_TCC`
 - `sf_zeta.tcc`, [637](#)
- `_M_omega_1`
 - `std::__detail::_jacobi_lattice_t`, [518](#)
- `_M_omega_3`
 - `std::__detail::_jacobi_lattice_t`, [518](#)
- `_N_FGH`
 - `std::__detail::_Airy_series`, [555](#)
- `_Num_Euler_Maclaurin_zeta`
 - `std::__detail`, [440](#)
- `_Num_Stieltjes`
 - `std::__detail`, [441](#)
- `_Real`
 - `std::__detail::_jacobi_lattice_t`, [514](#)
 - `std::__detail::_jacobi_theta_0_t`, [523](#)
 - `std::__detail::_weierstrass_invariants_t`, [528](#)
 - `std::__detail::_weierstrass_roots_t`, [531](#)

- std::__detail::__AiryState, [561](#)
- _Real_Omega1
 - std::__detail::__jacobi_lattice_t, [514](#)
- _Real_Omega3
 - std::__detail::__jacobi_lattice_t, [514](#)
- _S_Ai
 - std::__detail::__Airy_series, [551](#)
- _S_Ai0
 - std::__detail::__Airy_series, [555](#)
- _S_Aip0
 - std::__detail::__Airy_series, [555](#)
- _S_Airy
 - std::__detail::__Airy_series, [551](#)
- _S_Bi
 - std::__detail::__Airy_series, [552](#)
- _S_Bi0
 - std::__detail::__Airy_series, [555](#)
- _S_Bip0
 - std::__detail::__Airy_series, [556](#)
- _S_Euler_Maclaurin_zeta
 - std::__detail, [441](#)
- _S_FGH
 - std::__detail::__Airy_series, [552](#)
- _S_Fock
 - std::__detail::__Airy_series, [553](#)
- _S_Gi0
 - std::__detail::__Airy_series, [556](#)
- _S_Gip0
 - std::__detail::__Airy_series, [556](#)
- _S_Hi0
 - std::__detail::__Airy_series, [556](#)
- _S_Hip0
 - std::__detail::__Airy_series, [557](#)
- _S_Scorer
 - std::__detail::__Airy_series, [553](#)
- _S_Scorer2
 - std::__detail::__Airy_series, [554](#)
- _S_Stieltjes
 - std::__detail, [445](#)
- _S_absarg_ge_pio3
 - std::__detail::__Airy_asymp, [538](#)
- _S_absarg_lt_pio3
 - std::__detail::__Airy_asymp, [539](#)
- _S_c
 - std::__detail::__Airy_asymp_data< double >, [541](#)
 - std::__detail::__Airy_asymp_data< float >, [542](#)
 - std::__detail::__Airy_asymp_data< long double >, [543](#)
- _S_cheby
 - std::__detail::__gamma_lanczos_data< double >, [506](#)
 - std::__detail::__gamma_lanczos_data< float >, [507](#)
 - std::__detail::__gamma_lanczos_data< long double >, [508](#)
- std::__detail::__gamma_spouge_data< double >, [509](#)
- std::__detail::__gamma_spouge_data< float >, [510](#)
- std::__detail::__gamma_spouge_data< long double >, [511](#)
- _S_d
 - std::__detail::__Airy_asymp_data< double >, [541](#)
 - std::__detail::__Airy_asymp_data< float >, [542](#)
 - std::__detail::__Airy_asymp_data< long double >, [543](#)
- _S_double_factorial_table
 - std::__detail, [441](#)
- _S_eps
 - std::__detail::__Airy_series, [556](#)
- _S_factorial_table
 - std::__detail, [441](#)
- _S_g
 - std::__detail::__gamma_lanczos_data< double >, [506](#)
 - std::__detail::__gamma_lanczos_data< float >, [507](#)
 - std::__detail::__gamma_lanczos_data< long double >, [508](#)
- _S_harmonic_denom
 - std::__detail, [442](#)
- _S_harmonic_numer
 - std::__detail, [442](#)
- _S_i
 - std::__detail::__Airy_series, [557](#)
- _S_max_cd
 - std::__detail::__Airy_asymp_data< double >, [541](#)
 - std::__detail::__Airy_asymp_data< float >, [542](#)
 - std::__detail::__Airy_asymp_data< long double >, [543](#)
- _S_neg_double_factorial_table
 - std::__detail, [442](#)
- _S_num_double_factorials
 - std::__detail, [442](#)
- _S_num_double_factorials< double >
 - std::__detail, [442](#)
- _S_num_double_factorials< float >
 - std::__detail, [443](#)
- _S_num_double_factorials< long double >
 - std::__detail, [443](#)
- _S_num_factorials
 - std::__detail, [443](#)
- _S_num_factorials< double >
 - std::__detail, [443](#)
- _S_num_factorials< float >
 - std::__detail, [443](#)
- _S_num_factorials< long double >
 - std::__detail, [444](#)
- _S_num_harmonic_numer
 - std::__detail, [444](#)
- _S_num_neg_double_factorials

std::__detail, [444](#)
 _S_num_neg_double_factorials< double >
 std::__detail, [444](#)
 _S_num_neg_double_factorials< float >
 std::__detail, [444](#)
 _S_num_neg_double_factorials< long double >
 std::__detail, [445](#)
 _S_num_zetam1
 std::__detail, [445](#)
 _S_pi
 std::__detail::__jacobi_lattice_t, [518](#)
 std::__detail::__Airy_series, [557](#)
 _S_sqrt_pi
 std::__detail::__Airy_asymp_series, [546](#)
 std::__detail::__Airy_series, [557](#)
 _S_zetam1
 std::__detail, [445](#)
 _Terminator
 std::__detail::__Terminator, [566](#)
 _Tp_Nome
 std::__detail::__jacobi_lattice_t, [515](#)
 _Type
 std::__detail::__jacobi_theta_0_t, [523](#)
 std::__detail::__weierstrass_invariants_t, [528](#)
 std::__detail::__weierstrass_roots_t, [531](#)
 _Val
 std::__detail::__AiryAuxilliaryState, [558](#)
 __Ai_deriv
 __gnu_cxx::__airy_t, [448](#)
 std::__detail::__AiryState, [562](#)
 __Ai_value
 __gnu_cxx::__airy_t, [448](#)
 std::__detail::__AiryState, [562](#)
 __Bi_deriv
 __gnu_cxx::__airy_t, [448](#)
 std::__detail::__AiryState, [562](#)
 __Bi_value
 __gnu_cxx::__airy_t, [448](#)
 std::__detail::__AiryState, [562](#)
 __C_n
 __gnu_cxx::__gegenbauer_t, [477](#)
 __C_nm1
 __gnu_cxx::__gegenbauer_t, [477](#)
 __C_nm2
 __gnu_cxx::__gegenbauer_t, [478](#)
 __F_deriv
 __gnu_cxx::__cyl_coulomb_t, [461](#)
 __F_value
 __gnu_cxx::__cyl_coulomb_t, [461](#)
 __G_deriv
 __gnu_cxx::__cyl_coulomb_t, [462](#)
 __G_value
 __gnu_cxx::__cyl_coulomb_t, [462](#)
 __H1_deriv
 __gnu_cxx::__cyl_hankel_t, [464](#)
 __H1_value
 __gnu_cxx::__cyl_hankel_t, [464](#)
 __H2_deriv
 __gnu_cxx::__cyl_hankel_t, [464](#)
 __H2_value
 __gnu_cxx::__cyl_hankel_t, [464](#)
 __H_n
 __gnu_cxx::__hermite_t, [482](#)
 __H_nm1
 __gnu_cxx::__hermite_t, [482](#)
 __H_nm2
 __gnu_cxx::__hermite_t, [482](#)
 __He_n
 __gnu_cxx::__hermite_he_t, [480](#)
 __He_nm1
 __gnu_cxx::__hermite_he_t, [480](#)
 __He_nm2
 __gnu_cxx::__hermite_he_t, [480](#)
 __I_deriv
 __gnu_cxx::__cyl_mod_bessel_t, [466](#)
 __I_value
 __gnu_cxx::__cyl_mod_bessel_t, [467](#)
 __J_deriv
 __gnu_cxx::__cyl_bessel_t, [459](#)
 __J_value
 __gnu_cxx::__cyl_bessel_t, [459](#)
 __K_deriv
 __gnu_cxx::__cyl_mod_bessel_t, [467](#)
 __K_value
 __gnu_cxx::__cyl_mod_bessel_t, [467](#)
 __L_n
 __gnu_cxx::__laguerre_t, [491](#)
 __L_nm1
 __gnu_cxx::__laguerre_t, [491](#)
 __L_nm2
 __gnu_cxx::__laguerre_t, [491](#)
 __N_deriv
 __gnu_cxx::__cyl_bessel_t, [459](#)
 __N_value
 __gnu_cxx::__cyl_bessel_t, [459](#)
 __P_l
 __gnu_cxx::__legendre_p_t, [493](#)
 __P_lm1
 __gnu_cxx::__legendre_p_t, [493](#)
 __P_lm2
 __gnu_cxx::__legendre_p_t, [493](#)
 __P_n
 __gnu_cxx::__jacobi_t, [489](#)
 __P_nm1
 __gnu_cxx::__jacobi_t, [489](#)
 __P_nm2
 __gnu_cxx::__jacobi_t, [489](#)
 __STDCPP_MATH_SPEC_FUNCS__

specfun.h, [653](#)
 __T_n
 __gnu_cxx::__chebyshev_t_t, [450](#)
 __T_nm1
 __gnu_cxx::__chebyshev_t_t, [451](#)
 __T_nm2
 __gnu_cxx::__chebyshev_t_t, [451](#)
 __U_n
 __gnu_cxx::__chebyshev_u_t, [452](#)
 __U_nm1
 __gnu_cxx::__chebyshev_u_t, [453](#)
 __U_nm2
 __gnu_cxx::__chebyshev_u_t, [453](#)
 __V_n
 __gnu_cxx::__chebyshev_v_t, [454](#)
 __V_nm1
 __gnu_cxx::__chebyshev_v_t, [455](#)
 __V_nm2
 __gnu_cxx::__chebyshev_v_t, [455](#)
 __W_n
 __gnu_cxx::__chebyshev_w_t, [456](#)
 __W_nm1
 __gnu_cxx::__chebyshev_w_t, [457](#)
 __W_nm2
 __gnu_cxx::__chebyshev_w_t, [457](#)
 __Wronskian
 __gnu_cxx::__airy_t, [448](#)
 __gnu_cxx::__cyl_bessel_t, [458](#)
 __gnu_cxx::__cyl_coulomb_t, [461](#)
 __gnu_cxx::__cyl_hankel_t, [464](#)
 __gnu_cxx::__cyl_mod_bessel_t, [466](#)
 __gnu_cxx::__fock_airy_t, [469](#)
 __gnu_cxx::__sph_bessel_t, [498](#)
 __gnu_cxx::__sph_hankel_t, [501](#)
 __gnu_cxx::__sph_mod_bessel_t, [503](#)
 __airy
 std::__detail, [262](#)
 __airy_ai
 std::__detail, [262](#)
 __airy_arg
 std::__detail, [262](#)
 __airy_bi
 std::__detail, [263](#)
 __alpha1
 __gnu_cxx::__jacobi_t, [488](#)
 __gnu_cxx::__laguerre_t, [490](#)
 __am
 __gnu_cxx::__jacobi_ellint_t, [484](#)
 __assoc_laguerre
 std::__detail, [263](#)
 __assoc_legendre_p
 std::__detail, [264](#)
 __bernoulli
 std::__detail, [265](#)
 __bernoulli_2n
 std::__detail, [265](#)
 __bernoulli_series
 std::__detail, [266](#)
 __beta
 std::__detail, [267](#)
 __beta1
 __gnu_cxx::__jacobi_t, [488](#)
 __beta_gamma
 std::__detail, [267](#)
 __beta_inc
 std::__detail, [268](#)
 __beta_lgamma
 std::__detail, [269](#)
 __beta_p
 std::__detail, [269](#)
 __beta_product
 std::__detail, [269](#)
 __binomial
 std::__detail, [270](#), [271](#)
 __binomial_p
 std::__detail, [271](#)
 __binomial_pdf
 std::__detail, [272](#)
 __binomial_q
 std::__detail, [273](#)
 __bose_einstein
 std::__detail, [273](#)
 __cauchy_p
 std::__detail, [274](#)
 __cd
 __gnu_cxx::__jacobi_ellint_t, [484](#)
 __chebyshev_recur
 std::__detail, [274](#)
 __chebyshev_t
 std::__detail, [275](#)
 __chebyshev_u
 std::__detail, [275](#)
 __chebyshev_v
 std::__detail, [276](#)
 __chebyshev_w
 std::__detail, [277](#)
 __chi_squared_pdf
 std::__detail, [277](#)
 __chi_squared_pdfc
 std::__detail, [278](#)
 __chshint
 std::__detail, [278](#)
 __chshint_cont_frac
 std::__detail, [279](#)
 __chshint_series
 std::__detail, [279](#)
 __clamp_0_m2pi
 std::__detail, [279](#)

- `__clamp_pi`
 - `std::__detail`, [280](#)
- `__clausen`
 - `std::__detail`, [280](#), [281](#)
- `__clausen_cl`
 - `std::__detail`, [281](#), [282](#)
- `__clausen_sl`
 - `std::__detail`, [282](#), [283](#)
- `__cn_deriv`
 - `__gnu_cxx::__jacobi_ellint_t`, [484](#)
- `__cn_value`
 - `__gnu_cxx::__jacobi_ellint_t`, [486](#)
- `__comp_ellint_1`
 - `std::__detail`, [284](#)
- `__comp_ellint_2`
 - `std::__detail`, [284](#)
- `__comp_ellint_3`
 - `std::__detail`, [286](#)
- `__comp_ellint_d`
 - `std::__detail`, [286](#)
- `__comp_ellint_rf`
 - `std::__detail`, [287](#)
- `__comp_ellint_rg`
 - `std::__detail`, [287](#)
- `__conf_hyperg`
 - `std::__detail`, [287](#)
- `__conf_hyperg_lim`
 - `std::__detail`, [288](#)
- `__conf_hyperg_lim_series`
 - `std::__detail`, [288](#)
- `__conf_hyperg_luke`
 - `std::__detail`, [289](#)
- `__conf_hyperg_series`
 - `std::__detail`, [289](#)
- `__cos_pi`
 - `std::__detail`, [290](#)
- `__cos_v`
 - `__gnu_cxx::__sincos_t`, [497](#)
- `__cosh_pi`
 - `std::__detail`, [291](#)
- `__coshint`
 - `std::__detail`, [291](#)
- `__coulomb_CF1`
 - `std::__detail`, [292](#)
- `__coulomb_CF2`
 - `std::__detail`, [292](#)
- `__coulomb_f_recur`
 - `std::__detail`, [292](#)
- `__coulomb_g_recur`
 - `std::__detail`, [293](#)
- `__coulomb_norm`
 - `std::__detail`, [293](#)
- `__cpp_lib_math_special_functions`
 - `specfun.h`, [652](#)
- `__cs`
 - `__gnu_cxx::__jacobi_ellint_t`, [484](#)
- `__cyl_bessel`
 - `std::__detail`, [293](#)
- `__cyl_bessel_i`
 - `std::__detail`, [294](#)
- `__cyl_bessel_ij_series`
 - `std::__detail`, [295](#)
- `__cyl_bessel_ik`
 - `std::__detail`, [295](#)
- `__cyl_bessel_ik_asymp`
 - `std::__detail`, [296](#)
- `__cyl_bessel_ik_steel`
 - `std::__detail`, [297](#)
- `__cyl_bessel_j`
 - `std::__detail`, [297](#)
- `__cyl_bessel_jn`
 - `std::__detail`, [298](#)
- `__cyl_bessel_jn_asymp`
 - `std::__detail`, [298](#)
- `__cyl_bessel_jn_neg_arg`
 - `std::__detail`, [299](#)
- `__cyl_bessel_jn_steel`
 - `std::__detail`, [299](#)
- `__cyl_bessel_k`
 - `std::__detail`, [300](#)
- `__cyl_hankel_1`
 - `std::__detail`, [300](#), [301](#)
- `__cyl_hankel_2`
 - `std::__detail`, [302](#)
- `__cyl_neumann`
 - `std::__detail`, [303](#)
- `__cyl_neumann_n`
 - `std::__detail`, [303](#)
- `__dawson`
 - `std::__detail`, [304](#)
- `__dawson_cont_frac`
 - `std::__detail`, [304](#)
- `__dawson_series`
 - `std::__detail`, [305](#)
- `__dc`
 - `__gnu_cxx::__jacobi_ellint_t`, [484](#)
- `__debye`
 - `std::__detail`, [305](#)
- `__debye_region`
 - `std::__detail`, [306](#)
- `__delta`
 - `std::__detail::__weierstrass_invariants_t`, [529](#)
 - `std::__detail::__weierstrass_roots_t`, [532](#)
- `__digamma`
 - `std::__detail`, [306](#)
- `__digamma_asymp`
 - `std::__detail`, [307](#)
- `__digamma_series`

- std::__detail, 307
- __dilog
 - std::__detail, 308
- __dirichlet_beta
 - std::__detail, 308, 309
- __dirichlet_eta
 - std::__detail, 309, 310
- __dirichlet_lambda
 - std::__detail, 311
- __dn_value
 - __gnu_cxx::__jacobi_ellint_t, 486
- __double_factorial
 - std::__detail, 311
- __ds
 - __gnu_cxx::__jacobi_ellint_t, 485
- __e1
 - std::__detail::__weierstrass_roots_t, 533
- __e2
 - std::__detail::__weierstrass_roots_t, 533
- __e3
 - std::__detail::__weierstrass_roots_t, 533
- __ellint_1
 - std::__detail, 312
- __ellint_2
 - std::__detail, 312
- __ellint_3
 - std::__detail, 313
- __ellint_cel
 - std::__detail, 314
- __ellint_d
 - std::__detail, 314
- __ellint_el1
 - std::__detail, 314
- __ellint_el2
 - std::__detail, 314
- __ellint_el3
 - std::__detail, 315
- __ellint_rc
 - std::__detail, 315
- __ellint_rd
 - std::__detail, 316
- __ellint_rf
 - std::__detail, 317
- __ellint_rg
 - std::__detail, 317
- __ellint_rj
 - std::__detail, 318
- __ellnome
 - std::__detail, 319
 - std::__detail::__jacobi_lattice_t, 516
- __ellnome_k
 - std::__detail, 320
- __ellnome_series
 - std::__detail, 320
- __eta_arg
 - __gnu_cxx::__cyl_coulomb_t, 461
- __euler
 - std::__detail, 320, 321
- __euler_series
 - std::__detail, 321
- __eulerian_1
 - std::__detail, 321
- __eulerian_1_recur
 - std::__detail, 322
- __eulerian_2
 - std::__detail, 322
- __eulerian_2_recur
 - std::__detail, 322
- __exp2
 - std::__detail, 323
- __expint
 - std::__detail, 323, 324
- __expint_E1
 - std::__detail, 324
- __expint_E1_asymp
 - std::__detail, 325
- __expint_E1_series
 - std::__detail, 326
- __expint_Ei
 - std::__detail, 326
- __expint_Ei_asymp
 - std::__detail, 327
- __expint_Ei_series
 - std::__detail, 327
- __expint_En_asymp
 - std::__detail, 328
- __expint_En_cont_frac
 - std::__detail, 329
- __expint_En_large_n
 - std::__detail, 329
- __expint_En_recursion
 - std::__detail, 330
- __expint_En_series
 - std::__detail, 330
- __exponential_p
 - std::__detail, 331
- __exponential_pdf
 - std::__detail, 331
- __exponential_q
 - std::__detail, 332
- __factorial
 - std::__detail, 332
 - std::__detail::__Factorial_table, 565
- __fai_deriv
 - std::__detail::__AiryAuxilliaryState, 558
- __fai_value
 - std::__detail::__AiryAuxilliaryState, 559
- __falling_factorial

- std::__detail, 332, 333
- __fermi_dirac
 - std::__detail, 333
- __fisher_f_p
 - std::__detail, 334
- __fisher_f_pdf
 - std::__detail, 334
- __fisher_f_q
 - std::__detail, 335
- __fock_airy
 - std::__detail, 336
- __fp_is_equal
 - __gnu_cxx, 231
- __fp_is_even_integer
 - __gnu_cxx, 232
- __fp_is_half_integer
 - __gnu_cxx, 232
- __fp_is_half_odd_integer
 - __gnu_cxx, 233
- __fp_is_integer
 - __gnu_cxx, 233
- __fp_is_odd_integer
 - __gnu_cxx, 234
- __fp_is_zero
 - __gnu_cxx, 234
- __fp_max_abs
 - __gnu_cxx, 235
- __fresnel
 - std::__detail, 336
- __fresnel_cont_frac
 - std::__detail, 337
- __fresnel_series
 - std::__detail, 337
- __g_2
 - std::__detail::__weierstrass_invariants_t, 529
- __g_3
 - std::__detail::__weierstrass_invariants_t, 529
- __gai_deriv
 - std::__detail::__AiryAuxilliaryState, 559
- __gai_value
 - std::__detail::__AiryAuxilliaryState, 559
- __gamma
 - std::__detail, 337, 338
- __gamma_1_value
 - __gnu_cxx::__gamma_temme_t, 474
- __gamma_2_value
 - __gnu_cxx::__gamma_temme_t, 474
- __gamma_cont_frac
 - std::__detail, 338
- __gamma_minus_value
 - __gnu_cxx::__gamma_temme_t, 474
- __gamma_p
 - std::__detail, 338, 339
- __gamma_pdf
 - std::__detail, 339
- __gamma_plus_value
 - __gnu_cxx::__gamma_temme_t, 475
- __gamma_q
 - std::__detail, 340
- __gamma_reciprocal
 - std::__detail, 341
- __gamma_reciprocal_series
 - std::__detail, 341
- __gamma_series
 - std::__detail, 342
- __gamma_temme
 - std::__detail, 342
- __gappa_p_value
 - __gnu_cxx::__gappa_pq_t, 476
- __gappa_q_value
 - __gnu_cxx::__gappa_pq_t, 476
- __gauss
 - std::__detail, 343
- __gegenbauer_recur
 - std::__detail, 343
- __gegenbauer_zeros
 - std::__detail, 344
- __gnu_cxx, 217
 - __fp_is_equal, 231
 - __fp_is_even_integer, 232
 - __fp_is_half_integer, 232
 - __fp_is_half_odd_integer, 233
 - __fp_is_integer, 233
 - __fp_is_odd_integer, 234
 - __fp_is_zero, 234
 - __fp_max_abs, 235
 - __parity, 235
 - gauss_quad_type, 231
- __gnu_cxx::__airy_t
 - __Ai_deriv, 448
 - __Ai_value, 448
 - __Bi_deriv, 448
 - __Bi_value, 448
 - __Wronskian, 448
 - __x_arg, 449
- __gnu_cxx::__airy_t<_Tx, _Tp>, 447
- __gnu_cxx::__chebyshev_t_t
 - __T_n, 450
 - __T_nm1, 451
 - __T_nm2, 451
 - __n, 450
 - __x, 451
 - deriv, 450
 - deriv2, 450
- __gnu_cxx::__chebyshev_t_t<_Tp>, 449
- __gnu_cxx::__chebyshev_u_t
 - __U_n, 452
 - __U_nm1, 453

- __U_nm2, [453](#)
- __n, [452](#)
- __x, [453](#)
- deriv, [452](#)
- gnu_cxx::__chebyshev_u_t<_Tp>, [451](#)
- gnu_cxx::__chebyshev_v_t
 - __V_n, [454](#)
 - __V_nm1, [455](#)
 - __V_nm2, [455](#)
 - __n, [454](#)
 - __x, [455](#)
 - deriv, [454](#)
- gnu_cxx::__chebyshev_v_t<_Tp>, [453](#)
- gnu_cxx::__chebyshev_w_t
 - __W_n, [456](#)
 - __W_nm1, [457](#)
 - __W_nm2, [457](#)
 - __n, [456](#)
 - __x, [457](#)
 - deriv, [456](#)
- gnu_cxx::__chebyshev_w_t<_Tp>, [455](#)
- gnu_cxx::__cyl_bessel_t
 - __J_deriv, [459](#)
 - __J_value, [459](#)
 - __N_deriv, [459](#)
 - __N_value, [459](#)
 - __Wronskian, [458](#)
 - __nu_arg, [459](#)
 - __x_arg, [460](#)
- gnu_cxx::__cyl_bessel_t<_Tnu, _Tx, _Tp>, [457](#)
- gnu_cxx::__cyl_coulomb_t
 - __F_deriv, [461](#)
 - __F_value, [461](#)
 - __G_deriv, [462](#)
 - __G_value, [462](#)
 - __Wronskian, [461](#)
 - __eta_arg, [461](#)
 - __l, [462](#)
 - __rho_arg, [462](#)
- gnu_cxx::__cyl_coulomb_t<_Teta, _Trho, _Tp>, [460](#)
- gnu_cxx::__cyl_hankel_t
 - __H1_deriv, [464](#)
 - __H1_value, [464](#)
 - __H2_deriv, [464](#)
 - __H2_value, [464](#)
 - __Wronskian, [464](#)
 - __nu_arg, [465](#)
 - __x_arg, [465](#)
- gnu_cxx::__cyl_hankel_t<_Tnu, _Tx, _Tp>, [463](#)
- gnu_cxx::__cyl_mod_bessel_t
 - __I_deriv, [466](#)
 - __I_value, [467](#)
 - __K_deriv, [467](#)
 - __K_value, [467](#)
 - __Wronskian, [466](#)
 - __nu_arg, [467](#)
 - __x_arg, [468](#)
- gnu_cxx::__cyl_mod_bessel_t<_Tnu, _Tx, _Tp>, [465](#)
- gnu_cxx::__fock_airy_t
 - __Wronskian, [469](#)
 - __w1_deriv, [469](#)
 - __w1_value, [469](#)
 - __w2_deriv, [469](#)
 - __w2_value, [470](#)
 - __x_arg, [470](#)
- gnu_cxx::__fock_airy_t<_Tx, _Tp>, [468](#)
- gnu_cxx::__fp_is_integer_t, [470](#)
 - __is_integral, [471](#)
 - __value, [471](#)
 - operator bool, [471](#)
 - operator(), [471](#)
- gnu_cxx::__gamma_inc_t
 - __lgamma_value, [472](#)
 - __tgamma_value, [472](#)
- gnu_cxx::__gamma_inc_t<_Tp>, [472](#)
- gnu_cxx::__gamma_temme_t
 - __gamma_1_value, [474](#)
 - __gamma_2_value, [474](#)
 - __gamma_minus_value, [474](#)
 - __gamma_plus_value, [475](#)
 - __mu_arg, [475](#)
- gnu_cxx::__gamma_temme_t<_Tp>, [473](#)
- gnu_cxx::__gappa_pq_t
 - __gappa_p_value, [476](#)
 - __gappa_q_value, [476](#)
- gnu_cxx::__gappa_pq_t<_Tp>, [475](#)
- gnu_cxx::__gegenbauer_t
 - __C_n, [477](#)
 - __C_nm1, [477](#)
 - __C_nm2, [478](#)
 - __lambda, [478](#)
 - __n, [478](#)
 - __x, [478](#)
 - deriv, [477](#)
- gnu_cxx::__gegenbauer_t<_Tp>, [476](#)
- gnu_cxx::__hermite_he_t
 - __He_n, [480](#)
 - __He_nm1, [480](#)
 - __He_nm2, [480](#)
 - __n, [480](#)
 - __x, [480](#)
 - deriv, [479](#)
 - deriv2, [479](#)
- gnu_cxx::__hermite_he_t<_Tp>, [479](#)
- gnu_cxx::__hermite_t
 - __H_n, [482](#)
 - __H_nm1, [482](#)
 - __H_nm2, [482](#)

- __n, [482](#)
- __x, [483](#)
- deriv, [481](#)
- deriv2, [482](#)
- __gnu_cxx::__hermite_t< _Tp >, [481](#)
- __gnu_cxx::__jacobi_ellint_t
 - __am, [484](#)
 - __cd, [484](#)
 - __cn_deriv, [484](#)
 - __cn_value, [486](#)
 - __cs, [484](#)
 - __dc, [484](#)
 - __dn_value, [486](#)
 - __ds, [485](#)
 - __nc, [485](#)
 - __nd, [485](#)
 - __ns, [485](#)
 - __sc, [485](#)
 - __sd, [486](#)
 - __sn_deriv, [486](#)
 - __sn_value, [487](#)
- __gnu_cxx::__jacobi_ellint_t< _Tp >, [483](#)
- __gnu_cxx::__jacobi_t
 - __P_n, [489](#)
 - __P_nm1, [489](#)
 - __P_nm2, [489](#)
 - __alpha1, [488](#)
 - __beta1, [488](#)
 - __n, [488](#)
 - __x, [489](#)
 - deriv, [488](#)
- __gnu_cxx::__jacobi_t< _Tp >, [487](#)
- __gnu_cxx::__laguerre_t
 - __L_n, [491](#)
 - __L_nm1, [491](#)
 - __L_nm2, [491](#)
 - __alpha1, [490](#)
 - __n, [491](#)
 - __x, [491](#)
 - deriv, [490](#)
- __gnu_cxx::__laguerre_t< _Tpa, _Tp >, [490](#)
- __gnu_cxx::__legendre_p_t
 - __P_l, [493](#)
 - __P_lm1, [493](#)
 - __P_lm2, [493](#)
 - __l, [493](#)
 - __z, [493](#)
 - deriv, [492](#)
- __gnu_cxx::__legendre_p_t< _Tp >, [492](#)
- __gnu_cxx::__lgamma_t
 - __lgamma_sign, [494](#)
 - __lgamma_value, [494](#)
- __gnu_cxx::__lgamma_t< _Tp >, [494](#)
- __gnu_cxx::__quadrature_point_t
 - __point, [496](#)
 - __quadrature_point_t, [495](#), [496](#)
 - __weight, [496](#)
- __gnu_cxx::__quadrature_point_t< _Tp >, [495](#)
- __gnu_cxx::__sincos_t
 - __cos_v, [497](#)
 - __sin_v, [497](#)
- __gnu_cxx::__sincos_t< _Tp >, [496](#)
- __gnu_cxx::__sph_bessel_t
 - __Wronskian, [498](#)
 - __j_deriv, [499](#)
 - __j_value, [499](#)
 - __n_arg, [499](#)
 - __n_deriv, [499](#)
 - __n_value, [499](#)
 - __x_arg, [500](#)
- __gnu_cxx::__sph_bessel_t< _Tn, _Tx, _Tp >, [498](#)
- __gnu_cxx::__sph_hankel_t
 - __Wronskian, [501](#)
 - __h1_deriv, [501](#)
 - __h1_value, [501](#)
 - __h2_deriv, [501](#)
 - __h2_value, [502](#)
 - __n_arg, [502](#)
 - __x_arg, [502](#)
- __gnu_cxx::__sph_hankel_t< _Tn, _Tx, _Tp >, [500](#)
- __gnu_cxx::__sph_mod_bessel_t
 - __Wronskian, [503](#)
 - __i_deriv, [504](#)
 - __i_value, [504](#)
 - __k_deriv, [504](#)
 - __k_value, [504](#)
 - __n_arg, [504](#)
 - __x_arg, [505](#)
- __gnu_cxx::__sph_mod_bessel_t< _Tn, _Tx, _Tp >, [503](#)
- __h1_deriv
 - __gnu_cxx::__sph_hankel_t, [501](#)
- __h1_value
 - __gnu_cxx::__sph_hankel_t, [501](#)
- __h2_deriv
 - __gnu_cxx::__sph_hankel_t, [501](#)
- __h2_value
 - __gnu_cxx::__sph_hankel_t, [502](#)
- __hai_deriv
 - std::__detail::__AiryAuxilliaryState, [559](#)
- __hai_value
 - std::__detail::__AiryAuxilliaryState, [559](#)
- __hankel
 - std::__detail, [344](#)
- __hankel_debye
 - std::__detail, [345](#)
- __hankel_params
 - std::__detail, [345](#)
- __hankel_uniform

- std::__detail, 346
- __hankel_uniform_olver
 - std::__detail, 347
- __hankel_uniform_outer
 - std::__detail, 347
- __hankel_uniform_sum
 - std::__detail, 348
- __harmonic_number
 - std::__detail, 349
- __hermite
 - std::__detail, 349
- __hermite_asymp
 - std::__detail, 350
- __hermite_recur
 - std::__detail, 351
- __hermite_zeros
 - std::__detail, 352
- __heuman_lambda
 - std::__detail, 352
- __hurwitz_zeta
 - std::__detail, 352
- __hurwitz_zeta_euler_maclaurin
 - std::__detail, 353
- __hurwitz_zeta_polylog
 - std::__detail, 353
- __hydrogen
 - std::__detail, 354
- __hyperg
 - std::__detail, 354
- __hyperg_luke
 - std::__detail, 355
- __hyperg_recur
 - std::__detail, 355
- __hyperg_reflect
 - std::__detail, 356
- __hyperg_series
 - std::__detail, 357
- __i_deriv
 - __gnu_cxx::__sph_mod_bessel_t, 504
- __i_value
 - __gnu_cxx::__sph_mod_bessel_t, 504
- __ibeta_cont_frac
 - std::__detail, 358
- __is_integral
 - __gnu_cxx::__fp_is_integer_t, 471
- __j_deriv
 - __gnu_cxx::__sph_bessel_t, 499
- __j_value
 - __gnu_cxx::__sph_bessel_t, 499
- __jacobi_ellint
 - std::__detail, 358
- __jacobi_lattice_t
 - std::__detail::__jacobi_lattice_t, 515
- __jacobi_recur
 - std::__detail, 359
- __jacobi_theta_0_t
 - std::__detail::__jacobi_theta_0_t, 523
- __jacobi_theta_1
 - std::__detail, 359, 360
- __jacobi_theta_1_prod
 - std::__detail, 361
- __jacobi_theta_1_sum
 - std::__detail, 361
- __jacobi_theta_2
 - std::__detail, 362, 363
- __jacobi_theta_2_prod
 - std::__detail, 363
- __jacobi_theta_2_sum
 - std::__detail, 364
- __jacobi_theta_3
 - std::__detail, 365
- __jacobi_theta_3_prod
 - std::__detail, 366
- __jacobi_theta_3_sum
 - std::__detail, 367
- __jacobi_theta_4
 - std::__detail, 367, 368
- __jacobi_theta_4_prod
 - std::__detail, 369
- __jacobi_theta_4_sum
 - std::__detail, 369
- __jacobi_zeros
 - std::__detail, 370
- __jacobi_zeta
 - std::__detail, 370
- __k_deriv
 - __gnu_cxx::__sph_mod_bessel_t, 504
- __k_value
 - __gnu_cxx::__sph_mod_bessel_t, 504
- __klein_j
 - std::__detail::__weierstrass_invariants_t, 529
- __kolmogorov_p
 - std::__detail, 371
- __l
 - __gnu_cxx::__cyl_coulomb_t, 462
 - __gnu_cxx::__legendre_p_t, 493
- __laguerre
 - std::__detail, 371, 372
- __laguerre_hyperg
 - std::__detail, 372
- __laguerre_large_n
 - std::__detail, 373
- __laguerre_recur
 - std::__detail, 374
- __laguerre_zeros
 - std::__detail, 375
- __lambda
 - __gnu_cxx::__gegenbauer_t, 478

__lanczos_binet1p
 std::__detail, 375
 __lanczos_log_gamma1p
 std::__detail, 376
 __legendre_p
 std::__detail, 376
 __legendre_q
 std::__detail, 377
 __legendre_zeros
 std::__detail, 378
 __lgamma_sign
 gnu_cxx::__lgamma_t, 494
 __lgamma_value
 gnu_cxx::__gamma_inc_t, 472
 gnu_cxx::__lgamma_t, 494
 __log_binomial
 std::__detail, 378, 379
 __log_binomial_sign
 std::__detail, 379, 380
 __log_double_factorial
 std::__detail, 380
 __log_factorial
 std::__detail, 381
 std::__detail::Factorial_table, 565
 __log_falling_factorial
 std::__detail, 381
 __log_gamma
 std::__detail, 382, 383
 __log_gamma_bernoulli
 std::__detail, 383
 __log_gamma_sign
 std::__detail, 384
 __log_rising_factorial
 std::__detail, 384
 __log_stirling_1
 std::__detail, 385
 __log_stirling_1_sign
 std::__detail, 385
 __log_stirling_2
 std::__detail, 385
 __logint
 std::__detail, 386
 __logistic_p
 std::__detail, 386
 __logistic_pdf
 std::__detail, 387
 __lognormal_p
 std::__detail, 387
 __lognormal_pdf
 std::__detail, 387
 __m
 std::__detail::__jacobi_lattice_t::__arg_t, 519
 __max_FGH
 std::__detail, 440
 __max_FGH< double >
 std::__detail, 440
 __max_FGH< float >
 std::__detail, 440
 __mu_arg
 gnu_cxx::__gamma_temme_t, 475
 __n
 gnu_cxx::__chebyshev_t_t, 450
 gnu_cxx::__chebyshev_u_t, 452
 gnu_cxx::__chebyshev_v_t, 454
 gnu_cxx::__chebyshev_w_t, 456
 gnu_cxx::__gegenbauer_t, 478
 gnu_cxx::__hermite_he_t, 480
 gnu_cxx::__hermite_t, 482
 gnu_cxx::__jacobi_t, 488
 gnu_cxx::__laguerre_t, 491
 std::__detail::__jacobi_lattice_t::__arg_t, 520
 std::__detail::Factorial_table, 565
 __n_arg
 gnu_cxx::__sph_bessel_t, 499
 gnu_cxx::__sph_hankel_t, 502
 gnu_cxx::__sph_mod_bessel_t, 504
 __n_deriv
 gnu_cxx::__sph_bessel_t, 499
 __n_value
 gnu_cxx::__sph_bessel_t, 499
 __nc
 gnu_cxx::__jacobi_ellint_t, 485
 __nd
 gnu_cxx::__jacobi_ellint_t, 485
 __normal_p
 std::__detail, 388
 __normal_pdf
 std::__detail, 388
 __ns
 gnu_cxx::__jacobi_ellint_t, 485
 __nu_arg
 gnu_cxx::__cyl_bessel_t, 459
 gnu_cxx::__cyl_hankel_t, 465
 gnu_cxx::__cyl_mod_bessel_t, 467
 __omega_1
 std::__detail::__jacobi_lattice_t, 516
 __omega_2
 std::__detail::__jacobi_lattice_t, 516
 __omega_3
 std::__detail::__jacobi_lattice_t, 517
 __owens_t
 std::__detail, 388
 __parity
 gnu_cxx, 235
 __point
 gnu_cxx::__quadrature_point_t, 496
 __polar_pi
 std::__detail, 389

- __polygamma
 - std::__detail, 390
- __polylog
 - std::__detail, 390, 391
- __polylog_exp
 - std::__detail, 391
- __polylog_exp_asymp
 - std::__detail, 392
- __polylog_exp_neg
 - std::__detail, 393, 394
- __polylog_exp_neg_int
 - std::__detail, 394, 395
- __polylog_exp_neg_real
 - std::__detail, 395, 396
- __polylog_exp_pos
 - std::__detail, 397, 398
- __polylog_exp_pos_int
 - std::__detail, 399
- __polylog_exp_pos_real
 - std::__detail, 400, 401
- __polylog_exp_sum
 - std::__detail, 401
- __prob_hermite_recur
 - std::__detail, 402
- __quadrature_point_t
 - __gnu_cxx::__quadrature_point_t, 495, 496
- __radial_jacobi
 - std::__detail, 403
- __radial_jacobi_zeros
 - std::__detail, 404
- __reduce
 - std::__detail::__jacobi_lattice_t, 517
- __rho_arg
 - __gnu_cxx::__cyl_coulomb_t, 462
- __rice_pdf
 - std::__detail, 404
- __riemann_zeta
 - std::__detail, 405
- __riemann_zeta_euler_maclaurin
 - std::__detail, 405
- __riemann_zeta_glob
 - std::__detail, 406
- __riemann_zeta_laurent
 - std::__detail, 406
- __riemann_zeta_m_1
 - std::__detail, 406
- __riemann_zeta_m_1_glob
 - std::__detail, 407
- __riemann_zeta_product
 - std::__detail, 407
- __riemann_zeta_sum
 - std::__detail, 408
- __rising_factorial
 - std::__detail, 408, 409
- __sc
 - __gnu_cxx::__jacobi_ellint_t, 485
- __sd
 - __gnu_cxx::__jacobi_ellint_t, 486
- __sin_pi
 - std::__detail, 409, 410
- __sin_v
 - __gnu_cxx::__sincos_t, 497
- __sinc
 - std::__detail, 410
- __sinc_pi
 - std::__detail, 410
- __sincos
 - std::__detail, 411
- __sincos_pi
 - std::__detail, 412
- __sincosint
 - std::__detail, 412
- __sincosint_asymp
 - std::__detail, 412
- __sincosint_cont_frac
 - std::__detail, 413
- __sincosint_series
 - std::__detail, 413
- __sinh_pi
 - std::__detail, 413, 414
- __sinhc
 - std::__detail, 414
- __sinhc_pi
 - std::__detail, 414
- __sinhint
 - std::__detail, 415
- __sn_deriv
 - __gnu_cxx::__jacobi_ellint_t, 486
- __sn_value
 - __gnu_cxx::__jacobi_ellint_t, 487
- __sph_bessel
 - std::__detail, 415, 416
- __sph_bessel_ik
 - std::__detail, 417
- __sph_bessel_jn
 - std::__detail, 417
- __sph_bessel_jn_neg_arg
 - std::__detail, 418
- __sph_hankel
 - std::__detail, 418
- __sph_hankel_1
 - std::__detail, 419
- __sph_hankel_2
 - std::__detail, 420, 421
- __sph_harmonic
 - std::__detail, 421
- __sph_legendre
 - std::__detail, 422

- __sph_neumann
 - std::__detail, [423](#)
- __spouge_binet1p
 - std::__detail, [424](#)
- __spouge_log_gamma1p
 - std::__detail, [424](#)
- __stirling_1
 - std::__detail, [425](#)
- __stirling_1_recur
 - std::__detail, [426](#)
- __stirling_1_series
 - std::__detail, [426](#)
- __stirling_2
 - std::__detail, [427](#)
- __stirling_2_recur
 - std::__detail, [427](#)
- __stirling_2_series
 - std::__detail, [427](#)
- __student_t_p
 - std::__detail, [428](#)
- __student_t_pdf
 - std::__detail, [428](#)
- __student_t_q
 - std::__detail, [429](#)
- __tan_pi
 - std::__detail, [429](#), [430](#)
- __tanh_pi
 - std::__detail, [430](#)
- __tau
 - std::__detail::__jacobi_lattice_t, [518](#)
- __tau_t
 - std::__detail::__jacobi_lattice_t::__tau_t, [521](#)
- __tgamma
 - std::__detail, [431](#)
- __tgamma_lower
 - std::__detail, [431](#)
- __tgamma_value
 - gnu_cxx::__gamma_inc_t, [472](#)
- __theta_1
 - std::__detail, [431](#)
- __theta_2
 - std::__detail, [432](#)
- __theta_2_asymp
 - std::__detail, [433](#)
- __theta_2_sum
 - std::__detail, [433](#)
- __theta_3
 - std::__detail, [433](#)
- __theta_3_asymp
 - std::__detail, [434](#)
- __theta_3_sum
 - std::__detail, [434](#)
- __theta_4
 - std::__detail, [434](#)
- __theta_c
 - std::__detail, [435](#)
- __theta_d
 - std::__detail, [435](#)
- __theta_n
 - std::__detail, [436](#)
- __theta_s
 - std::__detail, [436](#)
- __tricoli_u
 - std::__detail, [436](#)
- __tricoli_u_naive
 - std::__detail, [437](#)
- __val
 - std::__detail::__jacobi_lattice_t::__tau_t, [521](#)
- __value
 - gnu_cxx::__fp_is_integer_t, [471](#)
- __w1_deriv
 - gnu_cxx::__fock_airy_t, [469](#)
- __w1_value
 - gnu_cxx::__fock_airy_t, [469](#)
- __w2_deriv
 - gnu_cxx::__fock_airy_t, [469](#)
- __w2_value
 - gnu_cxx::__fock_airy_t, [470](#)
- __weibull_p
 - std::__detail, [438](#)
- __weibull_pdf
 - std::__detail, [438](#)
- __weierstrass_invariants_t
 - std::__detail::__weierstrass_invariants_t, [528](#)
- __weierstrass_roots_t
 - std::__detail::__weierstrass_roots_t, [532](#)
- __weight
 - gnu_cxx::__quadrature_point_t, [496](#)
- __x
 - gnu_cxx::__chebyshev_t_t, [451](#)
 - gnu_cxx::__chebyshev_u_t, [453](#)
 - gnu_cxx::__chebyshev_v_t, [455](#)
 - gnu_cxx::__chebyshev_w_t, [457](#)
 - gnu_cxx::__gegenbauer_t, [478](#)
 - gnu_cxx::__hermite_he_t, [480](#)
 - gnu_cxx::__hermite_t, [483](#)
 - gnu_cxx::__jacobi_t, [489](#)
 - gnu_cxx::__laguerre_t, [491](#)
- __x_arg
 - gnu_cxx::__airy_t, [449](#)
 - gnu_cxx::__cyl_bessel_t, [460](#)
 - gnu_cxx::__cyl_hankel_t, [465](#)
 - gnu_cxx::__cyl_mod_bessel_t, [468](#)
 - gnu_cxx::__fock_airy_t, [470](#)
 - gnu_cxx::__sph_bessel_t, [500](#)
 - gnu_cxx::__sph_hankel_t, [502](#)
 - gnu_cxx::__sph_mod_bessel_t, [505](#)
- __z

- `__gnu_cxx::__legendre_p_t`, [493](#)
 - `std::__detail::__jacobi_lattice_t::__arg_t`, [520](#)
 - `std::__detail::__AiryAuxilliaryState`, [560](#)
 - `std::__detail::__AiryState`, [562](#)
- `__zernike`
 - `std::__detail`, [438](#)
- `__znorm1`
 - `std::__detail`, [439](#)
- `__znorm2`
 - `std::__detail`, [440](#)
- `airy_ai`
 - GNU Extended Mathematical Special Functions, [65](#)
- `airy_aif`
 - GNU Extended Mathematical Special Functions, [66](#)
- `airy_ail`
 - GNU Extended Mathematical Special Functions, [66](#)
- `airy_bi`
 - GNU Extended Mathematical Special Functions, [66](#), [67](#)
- `airy_bif`
 - GNU Extended Mathematical Special Functions, [67](#)
- `airy_bil`
 - GNU Extended Mathematical Special Functions, [68](#)
- `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, [25](#)
- `bernoulli`
 - GNU Extended Mathematical Special Functions, [68](#)
- `bernoullif`
 - GNU Extended Mathematical Special Functions, [69](#)
- `bernoullil`
 - GNU Extended Mathematical Special Functions, [69](#)
- `beta`
 - C++17/IS29124 Mathematical Special Functions, [25](#)
- `betaf`
 - C++17/IS29124 Mathematical Special Functions, [26](#)
- `betal`
 - C++17/IS29124 Mathematical Special Functions, [26](#)
- `binomial`
 - GNU Extended Mathematical Special Functions, [70](#)
- `binomial_p`
 - GNU Extended Mathematical Special Functions, [70](#)
- `binomial_pdf`
 - GNU Extended Mathematical Special Functions, [71](#)
- `binomialf`
 - GNU Extended Mathematical Special Functions, [72](#)
- `binomiall`
 - GNU Extended Mathematical Special Functions, [72](#)
- `bose_einstein`
 - GNU Extended Mathematical Special Functions, [72](#)
- `bose_einsteinf`
 - GNU Extended Mathematical Special Functions, [73](#)
- `bose_einsteinl`
 - GNU Extended Mathematical Special Functions, [73](#)
- 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`, [25](#)
 - `beta`, [25](#)
 - `betaf`, [26](#)
 - `betal`, [26](#)
 - `comp_ellint_1`, [26](#)
 - `comp_ellint_1f`, [27](#)
 - `comp_ellint_1l`, [27](#)
 - `comp_ellint_2`, [28](#)
 - `comp_ellint_2f`, [29](#)
 - `comp_ellint_2l`, [29](#)
 - `comp_ellint_3`, [29](#)
 - `comp_ellint_3f`, [30](#)
 - `comp_ellint_3l`, [30](#)
 - `cyl_bessel_i`, [30](#)
 - `cyl_bessel_if`, [31](#)
 - `cyl_bessel_il`, [31](#)
 - `cyl_bessel_j`, [32](#)
 - `cyl_bessel_jf`, [32](#)
 - `cyl_bessel_jl`, [33](#)
 - `cyl_bessel_k`, [33](#)
 - `cyl_bessel_kf`, [34](#)
 - `cyl_bessel_kl`, [34](#)
 - `cyl_neumann`, [34](#)
 - `cyl_neumannf`, [35](#)
 - `cyl_neumannl`, [35](#)
 - `ellint_1`, [36](#)
 - `ellint_1f`, [37](#)
 - `ellint_1l`, [37](#)
 - `ellint_2`, [37](#)
 - `ellint_2f`, [38](#)
 - `ellint_2l`, [38](#)
 - `ellint_3`, [38](#)
 - `ellint_3f`, [39](#)
 - `ellint_3l`, [40](#)
 - `expint`, [40](#)
 - `expintf`, [41](#)

- expintl, [41](#)
- hermite, [41](#)
- hermitef, [42](#)
- hermitel, [42](#)
- laguerre, [43](#)
- laguerref, [43](#)
- laguerrel, [44](#)
- legendre, [44](#)
- legendref, [45](#)
- legendrel, [45](#)
- riemann_zeta, [45](#)
- riemann_zetaf, [46](#)
- riemann_zetal, [46](#)
- sph_bessel, [47](#)
- sph_besself, [47](#)
- sph_bessell, [48](#)
- sph_legendre, [48](#)
- sph_legendref, [49](#)
- sph_legendrel, [49](#)
- sph_neumann, [49](#)
- sph_neumannf, [50](#)
- sph_neumannl, [50](#)
- chebyshev_t
 - GNU Extended Mathematical Special Functions, [73](#)
- chebyshev_tf
 - GNU Extended Mathematical Special Functions, [75](#)
- chebyshev_tl
 - GNU Extended Mathematical Special Functions, [75](#)
- chebyshev_u
 - GNU Extended Mathematical Special Functions, [75](#)
- chebyshev_uf
 - GNU Extended Mathematical Special Functions, [76](#)
- chebyshev_ul
 - GNU Extended Mathematical Special Functions, [76](#)
- chebyshev_v
 - GNU Extended Mathematical Special Functions, [77](#)
- chebyshev_vf
 - GNU Extended Mathematical Special Functions, [77](#)
- chebyshev_vl
 - GNU Extended Mathematical Special Functions, [78](#)
- chebyshev_w
 - GNU Extended Mathematical Special Functions, [78](#)
- chebyshev_wf
 - GNU Extended Mathematical Special Functions, [79](#)
- chebyshev_wl
 - GNU Extended Mathematical Special Functions, [79](#)
- clausen
 - GNU Extended Mathematical Special Functions, [79](#), [80](#)
- clausen_cl
 - GNU Extended Mathematical Special Functions, [81](#)
- clausen_clf
 - GNU Extended Mathematical Special Functions, [81](#)
- clausen_cll
 - GNU Extended Mathematical Special Functions, [82](#)
- clausen_sl
 - GNU Extended Mathematical Special Functions, [82](#)
- clausen_slf
 - GNU Extended Mathematical Special Functions, [83](#)
- clausen_sll
 - GNU Extended Mathematical Special Functions, [83](#)
- clausenf
 - GNU Extended Mathematical Special Functions, [83](#)
- clausenl
 - GNU Extended Mathematical Special Functions, [84](#)
- comp_ellint_1
 - C++17/IS29124 Mathematical Special Functions, [26](#)
- comp_ellint_1f
 - C++17/IS29124 Mathematical Special Functions, [27](#)
- comp_ellint_1l
 - C++17/IS29124 Mathematical Special Functions, [27](#)
- comp_ellint_2
 - C++17/IS29124 Mathematical Special Functions, [28](#)
- comp_ellint_2f
 - C++17/IS29124 Mathematical Special Functions, [29](#)
- comp_ellint_2l
 - C++17/IS29124 Mathematical Special Functions, [29](#)
- comp_ellint_3
 - C++17/IS29124 Mathematical Special Functions, [29](#)
- comp_ellint_3f
 - C++17/IS29124 Mathematical Special Functions, [30](#)
- comp_ellint_3l
 - C++17/IS29124 Mathematical Special Functions, [30](#)
- comp_ellint_d
 - GNU Extended Mathematical Special Functions, [84](#)
- comp_ellint_df
 - GNU Extended Mathematical Special Functions, [85](#)
- comp_ellint_dl
 - GNU Extended Mathematical Special Functions, [85](#)
- comp_ellint_rf
 - GNU Extended Mathematical Special Functions, [85](#), [86](#)
- comp_ellint_rg
 - GNU Extended Mathematical Special Functions, [87](#)
- conf_hyperg
 - GNU Extended Mathematical Special Functions, [88](#)
- conf_hyperg_lim
 - GNU Extended Mathematical Special Functions, [88](#)
- conf_hyperg_limf
 - GNU Extended Mathematical Special Functions, [89](#)
- conf_hyperg_liml
 - GNU Extended Mathematical Special Functions, [89](#)
- conf_hypergf
 - GNU Extended Mathematical Special Functions, [89](#)
- conf_hypergl
 - GNU Extended Mathematical Special Functions, [90](#)
- cos_pi
 - GNU Extended Mathematical Special Functions, [90](#)

- cos_pif
 - GNU Extended Mathematical Special Functions, [91](#)
- cos_pil
 - GNU Extended Mathematical Special Functions, [91](#)
- cosh_pi
 - GNU Extended Mathematical Special Functions, [91](#)
- cosh_pif
 - GNU Extended Mathematical Special Functions, [92](#)
- cosh_pil
 - GNU Extended Mathematical Special Functions, [92](#)
- coshint
 - GNU Extended Mathematical Special Functions, [92](#)
- coshintf
 - GNU Extended Mathematical Special Functions, [93](#)
- coshintl
 - GNU Extended Mathematical Special Functions, [93](#)
- cosint
 - GNU Extended Mathematical Special Functions, [93](#)
- cosintf
 - GNU Extended Mathematical Special Functions, [94](#)
- cosintl
 - GNU Extended Mathematical Special Functions, [94](#)
- cyl_bessel_i
 - C++17/IS29124 Mathematical Special Functions, [30](#)
- cyl_bessel_if
 - C++17/IS29124 Mathematical Special Functions, [31](#)
- cyl_bessel_il
 - C++17/IS29124 Mathematical Special Functions, [31](#)
- cyl_bessel_j
 - C++17/IS29124 Mathematical Special Functions, [32](#)
- cyl_bessel_jf
 - C++17/IS29124 Mathematical Special Functions, [32](#)
- cyl_bessel_jl
 - C++17/IS29124 Mathematical Special Functions, [33](#)
- cyl_bessel_k
 - C++17/IS29124 Mathematical Special Functions, [33](#)
- cyl_bessel_kf
 - C++17/IS29124 Mathematical Special Functions, [34](#)
- cyl_bessel_kl
 - C++17/IS29124 Mathematical Special Functions, [34](#)
- cyl_hankel_1
 - GNU Extended Mathematical Special Functions, [94](#), [95](#)
- cyl_hankel_1f
 - GNU Extended Mathematical Special Functions, [95](#), [96](#)
- cyl_hankel_1l
 - GNU Extended Mathematical Special Functions, [96](#)
- cyl_hankel_2
 - GNU Extended Mathematical Special Functions, [97](#)
- cyl_hankel_2f
 - GNU Extended Mathematical Special Functions, [98](#)
- cyl_hankel_2l
 - GNU Extended Mathematical Special Functions, [99](#)
- cyl_neumann
 - C++17/IS29124 Mathematical Special Functions, [34](#)
- cyl_neumannf
 - C++17/IS29124 Mathematical Special Functions, [35](#)
- cyl_neumannl
 - C++17/IS29124 Mathematical Special Functions, [35](#)
- dawson
 - GNU Extended Mathematical Special Functions, [99](#)
- dawsonf
 - GNU Extended Mathematical Special Functions, [100](#)
- dawsonl
 - GNU Extended Mathematical Special Functions, [100](#)
- debye
 - GNU Extended Mathematical Special Functions, [100](#)
- debyef
 - GNU Extended Mathematical Special Functions, [101](#)
- debyel
 - GNU Extended Mathematical Special Functions, [101](#)
- dedekind_eta
 - std::__detail::__jacobi_theta_0_t, [524](#)
- deriv
 - __gnu_cxx::__chebyshev_t_t, [450](#)
 - __gnu_cxx::__chebyshev_u_t, [452](#)
 - __gnu_cxx::__chebyshev_v_t, [454](#)
 - __gnu_cxx::__chebyshev_w_t, [456](#)
 - __gnu_cxx::__gegenbauer_t, [477](#)
 - __gnu_cxx::__hermite_he_t, [479](#)
 - __gnu_cxx::__hermite_t, [481](#)
 - __gnu_cxx::__jacobi_t, [488](#)
 - __gnu_cxx::__laguerre_t, [490](#)
 - __gnu_cxx::__legendre_p_t, [492](#)
- deriv2
 - __gnu_cxx::__chebyshev_t_t, [450](#)
 - __gnu_cxx::__hermite_he_t, [479](#)
 - __gnu_cxx::__hermite_t, [482](#)
- digamma
 - GNU Extended Mathematical Special Functions, [102](#)
- digammaf
 - GNU Extended Mathematical Special Functions, [102](#)
- digammal
 - GNU Extended Mathematical Special Functions, [103](#)
- dilog
 - GNU Extended Mathematical Special Functions, [103](#)
- dilogf
 - GNU Extended Mathematical Special Functions, [103](#)
- dilogl
 - GNU Extended Mathematical Special Functions, [104](#)
- dirichlet_beta
 - GNU Extended Mathematical Special Functions, [104](#)
- dirichlet_betaf
 - GNU Extended Mathematical Special Functions, [105](#)
- dirichlet_betal
 - GNU Extended Mathematical Special Functions, [105](#)

- dirichlet_eta
 - GNU Extended Mathematical Special Functions, [105](#)
- dirichlet_etaf
 - GNU Extended Mathematical Special Functions, [106](#)
- dirichlet_etal
 - GNU Extended Mathematical Special Functions, [106](#)
- dirichlet_lambda
 - GNU Extended Mathematical Special Functions, [106](#)
- dirichlet_lambdaf
 - GNU Extended Mathematical Special Functions, [107](#)
- dirichlet_lambdal
 - GNU Extended Mathematical Special Functions, [107](#)
- double_factorial
 - GNU Extended Mathematical Special Functions, [107](#)
- double_factorialf
 - GNU Extended Mathematical Special Functions, [107](#)
- double_factoriall
 - GNU Extended Mathematical Special Functions, [108](#)
- ellint_1
 - C++17/IS29124 Mathematical Special Functions, [36](#)
- ellint_1f
 - C++17/IS29124 Mathematical Special Functions, [37](#)
- ellint_1l
 - C++17/IS29124 Mathematical Special Functions, [37](#)
- ellint_2
 - C++17/IS29124 Mathematical Special Functions, [37](#)
- ellint_2f
 - C++17/IS29124 Mathematical Special Functions, [38](#)
- ellint_2l
 - C++17/IS29124 Mathematical Special Functions, [38](#)
- ellint_3
 - C++17/IS29124 Mathematical Special Functions, [38](#)
- ellint_3f
 - C++17/IS29124 Mathematical Special Functions, [39](#)
- ellint_3l
 - C++17/IS29124 Mathematical Special Functions, [40](#)
- ellint_cel
 - GNU Extended Mathematical Special Functions, [108](#)
- ellint_celf
 - GNU Extended Mathematical Special Functions, [109](#)
- ellint_cell
 - GNU Extended Mathematical Special Functions, [109](#)
- ellint_d
 - GNU Extended Mathematical Special Functions, [109](#)
- ellint_df
 - GNU Extended Mathematical Special Functions, [110](#)
- ellint_dl
 - GNU Extended Mathematical Special Functions, [110](#)
- ellint_el1
 - GNU Extended Mathematical Special Functions, [110](#)
- ellint_el1f
 - GNU Extended Mathematical Special Functions, [111](#)
- ellint_el1l
 - GNU Extended Mathematical Special Functions, [111](#)
- ellint_el2
 - GNU Extended Mathematical Special Functions, [111](#)
- ellint_el2f
 - GNU Extended Mathematical Special Functions, [112](#)
- ellint_el2l
 - GNU Extended Mathematical Special Functions, [112](#)
- ellint_el3
 - GNU Extended Mathematical Special Functions, [113](#)
- ellint_el3f
 - GNU Extended Mathematical Special Functions, [113](#)
- ellint_el3l
 - GNU Extended Mathematical Special Functions, [114](#)
- ellint_rc
 - GNU Extended Mathematical Special Functions, [114](#)
- ellint_rcf
 - GNU Extended Mathematical Special Functions, [115](#)
- ellint_rcl
 - GNU Extended Mathematical Special Functions, [115](#)
- ellint_rd
 - GNU Extended Mathematical Special Functions, [115](#)
- ellint_rdf
 - GNU Extended Mathematical Special Functions, [116](#)
- ellint_rdl
 - GNU Extended Mathematical Special Functions, [116](#)
- ellint_rf
 - GNU Extended Mathematical Special Functions, [117](#)
- ellint_rff
 - GNU Extended Mathematical Special Functions, [117](#)
- ellint_rfl
 - GNU Extended Mathematical Special Functions, [118](#)
- ellint_rg
 - GNU Extended Mathematical Special Functions, [118](#)
- ellint_rgf
 - GNU Extended Mathematical Special Functions, [119](#)
- ellint_rgl
 - GNU Extended Mathematical Special Functions, [119](#)
- ellint_rj
 - GNU Extended Mathematical Special Functions, [119](#)
- ellint_rjf
 - GNU Extended Mathematical Special Functions, [120](#)
- ellint_rjl
 - GNU Extended Mathematical Special Functions, [121](#)
- ellnome
 - GNU Extended Mathematical Special Functions, [121](#)
- ellnomef
 - GNU Extended Mathematical Special Functions, [122](#)
- ellnomel
 - GNU Extended Mathematical Special Functions, [122](#)
- eta_1
 - std::__detail::__jacobi_theta_0_t, [524](#)
- eta_2
 - std::__detail::__jacobi_theta_0_t, [524](#)
- eta_3

- std::__detail::__jacobi_theta_0_t, [525](#)
- euler
 - GNU Extended Mathematical Special Functions, [122](#)
- eulerian_1
 - GNU Extended Mathematical Special Functions, [123](#)
- eulerian_2
 - GNU Extended Mathematical Special Functions, [123](#)
- expint
 - C++17/IS29124 Mathematical Special Functions, [40](#)
 - GNU Extended Mathematical Special Functions, [123](#)
- expintf
 - C++17/IS29124 Mathematical Special Functions, [41](#)
 - GNU Extended Mathematical Special Functions, [124](#)
- expintl
 - C++17/IS29124 Mathematical Special Functions, [41](#)
 - GNU Extended Mathematical Special Functions, [124](#)
- exponential_p
 - GNU Extended Mathematical Special Functions, [125](#)
- exponential_pdf
 - GNU Extended Mathematical Special Functions, [125](#)
- factorial
 - GNU Extended Mathematical Special Functions, [125](#)
- factorialf
 - GNU Extended Mathematical Special Functions, [126](#)
- factoriall
 - GNU Extended Mathematical Special Functions, [126](#)
- falling_factorial
 - GNU Extended Mathematical Special Functions, [126](#)
- falling_factorialf
 - GNU Extended Mathematical Special Functions, [127](#)
- falling_factoriall
 - GNU Extended Mathematical Special Functions, [127](#)
- fermi_dirac
 - GNU Extended Mathematical Special Functions, [127](#)
- fermi_diracf
 - GNU Extended Mathematical Special Functions, [128](#)
- fermi_diracl
 - GNU Extended Mathematical Special Functions, [128](#)
- fisher_f_p
 - GNU Extended Mathematical Special Functions, [129](#)
- fisher_f_pdf
 - GNU Extended Mathematical Special Functions, [129](#)
- fresnel_c
 - GNU Extended Mathematical Special Functions, [130](#)
- fresnel_cf
 - GNU Extended Mathematical Special Functions, [130](#)
- fresnel_cl
 - GNU Extended Mathematical Special Functions, [131](#)
- fresnel_s
 - GNU Extended Mathematical Special Functions, [131](#)
- fresnel_sf
 - GNU Extended Mathematical Special Functions, [131](#)
- fresnel_sl
 - GNU Extended Mathematical Special Functions, [131](#)
- GNU Extended Mathematical Special Functions, [131](#)
- GNU Extended Mathematical Special Functions, [52](#)
 - airy_ai, [65](#)
 - airy_aif, [66](#)
 - airy_ail, [66](#)
 - airy_bi, [66](#), [67](#)
 - airy_bif, [67](#)
 - airy_bil, [68](#)
 - bernoulli, [68](#)
 - bernoullif, [69](#)
 - bernoullil, [69](#)
 - binomial, [70](#)
 - binomial_p, [70](#)
 - binomial_pdf, [71](#)
 - binomialf, [72](#)
 - binomiall, [72](#)
 - bose_einstein, [72](#)
 - bose_einsteinf, [73](#)
 - bose_einsteinl, [73](#)
 - chebyshev_t, [73](#)
 - chebyshev_tf, [75](#)
 - chebyshev_tl, [75](#)
 - chebyshev_u, [75](#)
 - chebyshev_uf, [76](#)
 - chebyshev_ul, [76](#)
 - chebyshev_v, [77](#)
 - chebyshev_vf, [77](#)
 - chebyshev_vl, [78](#)
 - chebyshev_w, [78](#)
 - chebyshev_wf, [79](#)
 - chebyshev_wl, [79](#)
 - clausen, [79](#), [80](#)
 - clausen_cl, [81](#)
 - clausen_clf, [81](#)
 - clausen_cll, [82](#)
 - clausen_sl, [82](#)
 - clausen_slf, [83](#)
 - clausen_sll, [83](#)
 - clausenf, [83](#)
 - clausenl, [84](#)
 - comp_ellint_d, [84](#)
 - comp_ellint_df, [85](#)
 - comp_ellint_dl, [85](#)
 - comp_ellint_rf, [85](#), [86](#)
 - comp_ellint_rg, [87](#)
 - conf_hyperg, [88](#)
 - conf_hyperg_lim, [88](#)
 - conf_hyperg_limf, [89](#)
 - conf_hyperg_liml, [89](#)
 - conf_hypergf, [89](#)
 - conf_hypergl, [90](#)
 - cos_pi, [90](#)
 - cos_pif, [91](#)

cos_pil, [91](#)
cosh_pi, [91](#)
cosh_pif, [92](#)
cosh_pil, [92](#)
coshint, [92](#)
coshintf, [93](#)
coshintl, [93](#)
cosint, [93](#)
cosintf, [94](#)
cosintl, [94](#)
cyl_hankel_1, [94](#), [95](#)
cyl_hankel_1f, [95](#), [96](#)
cyl_hankel_1l, [96](#)
cyl_hankel_2, [97](#)
cyl_hankel_2f, [98](#)
cyl_hankel_2l, [99](#)
dawson, [99](#)
dawsonf, [100](#)
dawsonl, [100](#)
debye, [100](#)
debyef, [101](#)
debyel, [101](#)
digamma, [102](#)
digammaf, [102](#)
digammal, [103](#)
dilog, [103](#)
dilogf, [103](#)
dilogl, [104](#)
dirichlet_beta, [104](#)
dirichlet_betaf, [105](#)
dirichlet_betel, [105](#)
dirichlet_eta, [105](#)
dirichlet_etaf, [106](#)
dirichlet_etel, [106](#)
dirichlet_lambda, [106](#)
dirichlet_lambdaf, [107](#)
dirichlet_lambdal, [107](#)
double_factorial, [107](#)
double_factorialf, [107](#)
double_factoriall, [108](#)
ellint_cel, [108](#)
ellint_celf, [109](#)
ellint_cell, [109](#)
ellint_d, [109](#)
ellint_df, [110](#)
ellint_dl, [110](#)
ellint_el1, [110](#)
ellint_el1f, [111](#)
ellint_el1l, [111](#)
ellint_el2, [111](#)
ellint_el2f, [112](#)
ellint_el2l, [112](#)
ellint_el3, [113](#)
ellint_el3f, [113](#)
ellint_el3l, [114](#)
ellint_rc, [114](#)
ellint_rcf, [115](#)
ellint_rcl, [115](#)
ellint_rd, [115](#)
ellint_rdf, [116](#)
ellint_rdl, [116](#)
ellint_rf, [117](#)
ellint_rff, [117](#)
ellint_rfl, [118](#)
ellint_rg, [118](#)
ellint_rgf, [119](#)
ellint_rgl, [119](#)
ellint_rj, [119](#)
ellint_rjf, [120](#)
ellint_rjl, [121](#)
ellnome, [121](#)
ellnomef, [122](#)
ellnomel, [122](#)
euler, [122](#)
eulerian_1, [123](#)
eulerian_2, [123](#)
expint, [123](#)
expintf, [124](#)
expintl, [124](#)
exponential_p, [125](#)
exponential_pdf, [125](#)
factorial, [125](#)
factorialf, [126](#)
factoriall, [126](#)
falling_factorial, [126](#)
falling_factorialf, [127](#)
falling_factoriall, [127](#)
fermi_dirac, [127](#)
fermi_diracf, [128](#)
fermi_diracl, [128](#)
fisher_f_p, [129](#)
fisher_f_pdf, [129](#)
fresnel_c, [130](#)
fresnel_cf, [130](#)
fresnel_cl, [131](#)
fresnel_s, [131](#)
fresnel_sf, [131](#)
fresnel_sl, [131](#)
gamma_p, [132](#)
gamma_pdf, [132](#)
gamma_pf, [132](#)
gamma_pl, [133](#)
gamma_q, [133](#)
gamma_qf, [133](#)
gamma_ql, [133](#)
gamma_reciprocal, [134](#)
gamma_reciprocalf, [134](#)
gamma_reciprocall, [134](#)

gegenbauer, 135
gegenbauerf, 135
gegenbauerl, 136
harmonic, 136
heuman_lambda, 137
heuman_lambdaf, 137
heuman_lambdal, 137
hurwitz_zeta, 138
hurwitz_zetaf, 138
hurwitz_zetal, 139
hyperg, 139
hypergf, 140
hypergl, 140
ibeta, 140
ibetac, 141
ibetacf, 142
ibetacl, 142
ibetaf, 142
ibetal, 142
jacobi, 143
jacobi_cn, 144
jacobi_cnf, 144
jacobi_cnl, 145
jacobi_dn, 145
jacobi_dnf, 146
jacobi_dnl, 146
jacobi_sn, 146
jacobi_snf, 147
jacobi_snl, 147
jacobi_theta_1, 148
jacobi_theta_1f, 148
jacobi_theta_1l, 149
jacobi_theta_2, 149
jacobi_theta_2f, 150
jacobi_theta_2l, 150
jacobi_theta_3, 150
jacobi_theta_3f, 151
jacobi_theta_3l, 151
jacobi_theta_4, 151
jacobi_theta_4f, 152
jacobi_theta_4l, 152
jacobi_zeta, 152
jacobi_zetaf, 153
jacobi_zetal, 153
jacobif, 153
jacobil, 153
lbinomial, 154
lbinomialf, 155
lbinomiall, 155
ldouble_factorial, 155
ldouble_factorialf, 155
ldouble_factoriall, 156
legendre_q, 156
legendre_qf, 157
legendre_ql, 157
lerch_phi, 157
lerch_phif, 158
lerch_phil, 158
lfactorial, 158
lfactorialf, 159
lfactoriall, 159
lfalling_factorial, 159
lfalling_factorialf, 160
lfalling_factoriall, 160
lgamma, 160, 161
lgammaf, 161
lgammal, 162
logint, 162
logintf, 163
logintl, 163
logistic_p, 163
logistic_pdf, 164
lognormal_p, 164
lognormal_pdf, 164
lrising_factorial, 165
lrising_factorialf, 165
lrising_factoriall, 165
normal_p, 166
normal_pdf, 166
owens_t, 166
owens_tf, 167
owens_tl, 167
polygamma, 167
polygammaf, 168
polygammal, 168
polylog, 168, 169
polylogf, 169, 170
polylogl, 170
radpoly, 171
radpolyf, 172
radpolyl, 172
rising_factorial, 172
rising_factorialf, 173
rising_factoriall, 173
sin_pi, 173
sin_pif, 174
sin_pil, 174
sinc, 174
sinc_pi, 175
sinc_pif, 175
sinc_pil, 176
sincf, 176
sincl, 176
sincos, 177
sincos_pi, 177
sincos_pif, 178
sincos_pil, 178
sincosf, 178

- sincosl, [179](#)
- sinh_pi, [179](#)
- sinh_pif, [180](#)
- sinh_pil, [180](#)
- sinhc, [180](#)
- sinhc_pi, [181](#)
- sinhc_pif, [181](#)
- sinhc_pil, [182](#)
- sinhcf, [182](#)
- sinhcl, [182](#)
- sinhint, [182](#)
- sinhintf, [183](#)
- sinhintl, [183](#)
- sinint, [183](#)
- sinintf, [184](#)
- sinintl, [184](#)
- sph_bessel_i, [184](#)
- sph_bessel_if, [185](#)
- sph_bessel_il, [185](#)
- sph_bessel_k, [186](#)
- sph_bessel_kf, [187](#)
- sph_bessel_kl, [187](#)
- sph_hankel_1, [187](#), [188](#)
- sph_hankel_1f, [188](#), [189](#)
- sph_hankel_1l, [189](#)
- sph_hankel_2, [190](#), [191](#)
- sph_hankel_2f, [191](#)
- sph_hankel_2l, [192](#)
- sph_harmonic, [192](#)
- sph_harmonicf, [193](#)
- sph_harmonicl, [193](#)
- stirling_1, [194](#)
- stirling_2, [194](#)
- student_t_p, [195](#)
- student_t_pdf, [195](#)
- tan_pi, [196](#)
- tan_pif, [196](#)
- tan_pil, [197](#)
- tanh_pi, [197](#)
- tanh_pif, [198](#)
- tanh_pil, [198](#)
- tgamma, [198](#), [199](#)
- tgamma_lower, [199](#)
- tgamma_lowerf, [199](#)
- tgamma_lowerl, [200](#)
- tgammaf, [200](#), [201](#)
- tgammal, [201](#), [202](#)
- theta_1, [202](#)
- theta_1f, [203](#)
- theta_1l, [203](#)
- theta_2, [203](#)
- theta_2f, [204](#)
- theta_2l, [204](#)
- theta_3, [204](#)
- theta_3f, [205](#)
- theta_3l, [205](#)
- theta_4, [205](#)
- theta_4f, [206](#)
- theta_4l, [206](#)
- theta_c, [206](#)
- theta_cf, [207](#)
- theta_cl, [207](#)
- theta_d, [208](#)
- theta_df, [208](#)
- theta_dl, [209](#)
- theta_n, [209](#)
- theta_nf, [210](#)
- theta_nl, [210](#)
- theta_s, [210](#)
- theta_sf, [211](#)
- theta_sl, [211](#)
- tricoli_u, [212](#)
- tricoli_uf, [213](#)
- tricoli_ul, [213](#)
- weibull_p, [213](#)
- weibull_pdf, [213](#)
- zernike, [214](#)
- zernikef, [215](#)
- zernikel, [215](#)
- gamma_p
 - GNU Extended Mathematical Special Functions, [132](#)
- gamma_pdf
 - GNU Extended Mathematical Special Functions, [132](#)
- gamma_pf
 - GNU Extended Mathematical Special Functions, [132](#)
- gamma_pl
 - GNU Extended Mathematical Special Functions, [133](#)
- gamma_q
 - GNU Extended Mathematical Special Functions, [133](#)
- gamma_qf
 - GNU Extended Mathematical Special Functions, [133](#)
- gamma_ql
 - GNU Extended Mathematical Special Functions, [133](#)
- gamma_reciprocal
 - GNU Extended Mathematical Special Functions, [134](#)
- gamma_reciprocalf
 - GNU Extended Mathematical Special Functions, [134](#)
- gamma_reciprocall
 - GNU Extended Mathematical Special Functions, [134](#)
- gauss_quad_type
 - __gnu_cxx, [231](#)
- gegenbauer
 - GNU Extended Mathematical Special Functions, [135](#)
- gegenbauerf
 - GNU Extended Mathematical Special Functions, [135](#)
- gegenbauerl
 - GNU Extended Mathematical Special Functions, [136](#)

- harmonic
 - GNU Extended Mathematical Special Functions, [136](#)
- hermite
 - C++17/IS29124 Mathematical Special Functions, [41](#)
- hermitef
 - C++17/IS29124 Mathematical Special Functions, [42](#)
- hermitel
 - C++17/IS29124 Mathematical Special Functions, [42](#)
- heuman_lambda
 - GNU Extended Mathematical Special Functions, [137](#)
- heuman_lambdaf
 - GNU Extended Mathematical Special Functions, [137](#)
- heuman_lambdal
 - GNU Extended Mathematical Special Functions, [137](#)
- hurwitz_zeta
 - GNU Extended Mathematical Special Functions, [138](#)
- hurwitz_zetaf
 - GNU Extended Mathematical Special Functions, [138](#)
- hurwitz_zetal
 - GNU Extended Mathematical Special Functions, [139](#)
- hyperg
 - GNU Extended Mathematical Special Functions, [139](#)
- hypergf
 - GNU Extended Mathematical Special Functions, [140](#)
- hypergl
 - GNU Extended Mathematical Special Functions, [140](#)
- ibeta
 - GNU Extended Mathematical Special Functions, [140](#)
- ibetac
 - GNU Extended Mathematical Special Functions, [141](#)
- ibetacf
 - GNU Extended Mathematical Special Functions, [142](#)
- ibetac1
 - GNU Extended Mathematical Special Functions, [142](#)
- ibetaf
 - GNU Extended Mathematical Special Functions, [142](#)
- ibetal
 - GNU Extended Mathematical Special Functions, [142](#)
- include/bits/sf_airy.tcc, [569](#)
- include/bits/sf_bernoulli.tcc, [571](#)
- include/bits/sf_bessel.tcc, [572](#)
- include/bits/sf_beta.tcc, [575](#)
- include/bits/sf_cardinal.tcc, [577](#)
- include/bits/sf_chebyshev.tcc, [579](#)
- include/bits/sf_coulomb.tcc, [580](#)
- include/bits/sf_dawson.tcc, [582](#)
- include/bits/sf_distributions.tcc, [584](#)
- include/bits/sf_ellint.tcc, [587](#)
- include/bits/sf_euler.tcc, [590](#)
- include/bits/sf_expint.tcc, [591](#)
- include/bits/sf_fresnel.tcc, [594](#)
- include/bits/sf_gamma.tcc, [595](#)
- include/bits/sf_gegenbauer.tcc, [603](#)
- include/bits/sf_hankel.tcc, [605](#)
- include/bits/sf_hermite.tcc, [608](#)
- include/bits/sf_hyperg.tcc, [609](#)
- include/bits/sf_hypint.tcc, [612](#)
- include/bits/sf_jacobi.tcc, [614](#)
- include/bits/sf_laguerre.tcc, [615](#)
- include/bits/sf_legendre.tcc, [617](#)
- include/bits/sf_mod_bessel.tcc, [619](#)
- include/bits/sf_owens_t.tcc, [621](#)
- include/bits/sf_polylog.tcc, [623](#)
- include/bits/sf_stirling.tcc, [626](#)
- include/bits/sf_theta.tcc, [627](#)
- include/bits/sf_trig.tcc, [631](#)
- include/bits/sf_trigint.tcc, [633](#)
- include/bits/sf_zeta.tcc, [635](#)
- include/bits/specfun.h, [637](#)
- include/bits/specfun_state.h, [653](#)
- include/ext/math_util.h, [655](#)
- inner_radius
 - std::__detail::__Airy, [536](#)
 - std::__detail::__Airy_default_radii< double >, [547](#)
 - std::__detail::__Airy_default_radii< float >, [548](#)
 - std::__detail::__Airy_default_radii< long double >, [549](#)
- jacobi
 - GNU Extended Mathematical Special Functions, [143](#)
- jacobi_cn
 - GNU Extended Mathematical Special Functions, [144](#)
- jacobi_cnf
 - GNU Extended Mathematical Special Functions, [144](#)
- jacobi_cnl
 - GNU Extended Mathematical Special Functions, [145](#)
- jacobi_dn
 - GNU Extended Mathematical Special Functions, [145](#)
- jacobi_dnf
 - GNU Extended Mathematical Special Functions, [146](#)
- jacobi_dnl
 - GNU Extended Mathematical Special Functions, [146](#)
- jacobi_sn
 - GNU Extended Mathematical Special Functions, [146](#)
- jacobi_snf
 - GNU Extended Mathematical Special Functions, [147](#)
- jacobi_snl
 - GNU Extended Mathematical Special Functions, [147](#)
- jacobi_theta_1
 - GNU Extended Mathematical Special Functions, [148](#)
- jacobi_theta_1f
 - GNU Extended Mathematical Special Functions, [148](#)
- jacobi_theta_1l
 - GNU Extended Mathematical Special Functions, [149](#)
- jacobi_theta_2
 - GNU Extended Mathematical Special Functions, [149](#)
- jacobi_theta_2f

- GNU Extended Mathematical Special Functions, [150](#)
- `jacobi_theta_2l`
 - GNU Extended Mathematical Special Functions, [150](#)
- `jacobi_theta_3`
 - GNU Extended Mathematical Special Functions, [150](#)
- `jacobi_theta_3f`
 - GNU Extended Mathematical Special Functions, [151](#)
- `jacobi_theta_3l`
 - GNU Extended Mathematical Special Functions, [151](#)
- `jacobi_theta_4`
 - GNU Extended Mathematical Special Functions, [151](#)
- `jacobi_theta_4f`
 - GNU Extended Mathematical Special Functions, [152](#)
- `jacobi_theta_4l`
 - GNU Extended Mathematical Special Functions, [152](#)
- `jacobi_zeta`
 - GNU Extended Mathematical Special Functions, [152](#)
- `jacobi_zetaf`
 - GNU Extended Mathematical Special Functions, [153](#)
- `jacobi_zetal`
 - GNU Extended Mathematical Special Functions, [153](#)
- `jacobif`
 - GNU Extended Mathematical Special Functions, [153](#)
- `jacobil`
 - GNU Extended Mathematical Special Functions, [153](#)
- `laguerre`
 - C++17/IS29124 Mathematical Special Functions, [43](#)
- `laguerref`
 - C++17/IS29124 Mathematical Special Functions, [43](#)
- `laguerrel`
 - C++17/IS29124 Mathematical Special Functions, [44](#)
- `lbinomial`
 - GNU Extended Mathematical Special Functions, [154](#)
- `lbinomialf`
 - GNU Extended Mathematical Special Functions, [155](#)
- `lbinomiall`
 - GNU Extended Mathematical Special Functions, [155](#)
- `ldouble_factorial`
 - GNU Extended Mathematical Special Functions, [155](#)
- `ldouble_factorialf`
 - GNU Extended Mathematical Special Functions, [155](#)
- `ldouble_factoriall`
 - GNU Extended Mathematical Special Functions, [156](#)
- `legendre`
 - C++17/IS29124 Mathematical Special Functions, [44](#)
- `legendre_q`
 - GNU Extended Mathematical Special Functions, [156](#)
- `legendre_qf`
 - GNU Extended Mathematical Special Functions, [157](#)
- `legendre_ql`
 - GNU Extended Mathematical Special Functions, [157](#)
- `legendref`
 - C++17/IS29124 Mathematical Special Functions, [45](#)
- `legendrel`
 - C++17/IS29124 Mathematical Special Functions, [45](#)
- `lerch_phi`
 - GNU Extended Mathematical Special Functions, [157](#)
- `lerch_phif`
 - GNU Extended Mathematical Special Functions, [158](#)
- `lerch_phil`
 - GNU Extended Mathematical Special Functions, [158](#)
- `lfactorial`
 - GNU Extended Mathematical Special Functions, [158](#)
- `lfactorialf`
 - GNU Extended Mathematical Special Functions, [159](#)
- `lfactoriall`
 - GNU Extended Mathematical Special Functions, [159](#)
- `lfalling_factorial`
 - GNU Extended Mathematical Special Functions, [159](#)
- `lfalling_factorialf`
 - GNU Extended Mathematical Special Functions, [160](#)
- `lfalling_factoriall`
 - GNU Extended Mathematical Special Functions, [160](#)
- `lgamma`
 - GNU Extended Mathematical Special Functions, [160](#), [161](#)
- `lgammaf`
 - GNU Extended Mathematical Special Functions, [161](#)
- `lgammal`
 - GNU Extended Mathematical Special Functions, [162](#)
- `logint`
 - GNU Extended Mathematical Special Functions, [162](#)
- `logintf`
 - GNU Extended Mathematical Special Functions, [163](#)
- `logintl`
 - GNU Extended Mathematical Special Functions, [163](#)
- `logistic_p`
 - GNU Extended Mathematical Special Functions, [163](#)
- `logistic_pdf`
 - GNU Extended Mathematical Special Functions, [164](#)
- `lognormal_p`
 - GNU Extended Mathematical Special Functions, [164](#)
- `lognormal_pdf`
 - GNU Extended Mathematical Special Functions, [164](#)
- `lrising_factorial`
 - GNU Extended Mathematical Special Functions, [165](#)
- `lrising_factorialf`
 - GNU Extended Mathematical Special Functions, [165](#)
- `lrising_factoriall`
 - GNU Extended Mathematical Special Functions, [165](#)
- `normal_p`
 - GNU Extended Mathematical Special Functions, [166](#)
- `normal_pdf`
 - GNU Extended Mathematical Special Functions, [166](#)
- `num_terms`
 - `std::__detail::_AsympTerminator`, [564](#)

- std::__detail::Terminator, [567](#)
- operator bool
 - __gnu_cxx::__fp_is_integer_t, [471](#)
- operator<<
 - std::__detail::AsympTerminator, [564](#)
- operator()
 - __gnu_cxx::__fp_is_integer_t, [471](#)
 - std::__detail::Airy, [535](#)
 - std::__detail::Airy_asymp, [539](#)
 - std::__detail::Airy_asymp_series, [546](#)
 - std::__detail::AsympTerminator, [564](#)
 - std::__detail::Terminator, [567](#)
- outer_radius
 - std::__detail::Airy, [536](#)
 - std::__detail::Airy_default_radii< double >, [547](#)
 - std::__detail::Airy_default_radii< float >, [548](#)
 - std::__detail::Airy_default_radii< long double >, [549](#)
- owens_t
 - GNU Extended Mathematical Special Functions, [166](#)
- owens_tf
 - GNU Extended Mathematical Special Functions, [167](#)
- owens_tl
 - GNU Extended Mathematical Special Functions, [167](#)
- polygamma
 - GNU Extended Mathematical Special Functions, [167](#)
- polygammaf
 - GNU Extended Mathematical Special Functions, [168](#)
- polygamma1
 - GNU Extended Mathematical Special Functions, [168](#)
- polylog
 - GNU Extended Mathematical Special Functions, [168](#), [169](#)
- polylogf
 - GNU Extended Mathematical Special Functions, [169](#), [170](#)
- polylog1
 - GNU Extended Mathematical Special Functions, [170](#)
- radpoly
 - GNU Extended Mathematical Special Functions, [171](#)
- radpolyf
 - GNU Extended Mathematical Special Functions, [172](#)
- radpoly1
 - GNU Extended Mathematical Special Functions, [172](#)
- riemann_zeta
 - C++17/IS29124 Mathematical Special Functions, [45](#)
- riemann_zetaf
 - C++17/IS29124 Mathematical Special Functions, [46](#)
- riemann_zetal
 - C++17/IS29124 Mathematical Special Functions, [46](#)
- rising_factorial
 - GNU Extended Mathematical Special Functions, [172](#)
- rising_factorialf
 - GNU Extended Mathematical Special Functions, [173](#)
- rising_factoriall
 - GNU Extended Mathematical Special Functions, [173](#)
- scalar_type
 - std::__detail::Airy, [534](#)
 - std::__detail::Airy_asymp_series, [545](#)
- sf_airy.tcc
 - _GLIBCXX_BITS_SF_AIRY_TCC, [571](#)
- sf_bernoulli.tcc
 - _GLIBCXX_BITS_SF_BERNOULLI_TCC, [572](#)
- sf_bessel.tcc
 - _GLIBCXX_BITS_SF_BESSEL_TCC, [575](#)
- sf_beta.tcc
 - _GLIBCXX_BITS_SF_BETA_TCC, [576](#)
- sf_cardinal.tcc
 - _GLIBCXX_BITS_SF_CARDINAL_TCC, [578](#)
- sf_chebyshev.tcc
 - _GLIBCXX_BITS_SF_CHEBYSHEV_TCC, [580](#)
- sf_coulomb.tcc
 - _GLIBCXX_BITS_SF_COULOMB_TCC, [582](#)
- sf_dawson.tcc
 - _GLIBCXX_BITS_SF_DAWSON_TCC, [583](#)
- sf_distributions.tcc
 - _GLIBCXX_BITS_SF_DISTRIBUTIONS_TCC, [587](#)
- sf_ellint.tcc
 - _GLIBCXX_BITS_SF_ELLINT_TCC, [589](#)
- sf_euler.tcc
 - _GLIBCXX_BITS_SF_EULER_TCC, [591](#)
- sf_expint.tcc
 - _GLIBCXX_BITS_SF_EXPINT_TCC, [593](#)
- sf_fresnel.tcc
 - _GLIBCXX_BITS_SF_FRESNEL_TCC, [595](#)
- sf_gamma.tcc
 - _GLIBCXX_BITS_SF_GAMMA_TCC, [603](#)
- sf_gegenbauer.tcc
 - _GLIBCXX_BITS_SF_GEGENBAUER_TCC, [604](#)
- sf_hankel.tcc
 - _GLIBCXX_BITS_SF_HANKEL_TCC, [607](#)
- sf_hermite.tcc
 - _GLIBCXX_BITS_SF_HERMITE_TCC, [609](#)
- sf_hypertcc
 - _GLIBCXX_BITS_SF_HYPERG_TCC, [612](#)
- sf_hypint.tcc
 - _GLIBCXX_BITS_SF_HYPINT_TCC, [613](#)
- sf_jacobi.tcc
 - _GLIBCXX_BITS_SF_JACOBI_TCC, [615](#)
- sf_laguerre.tcc
 - _GLIBCXX_BITS_SF_LAGUERRE_TCC, [617](#)
- sf_legendre.tcc
 - _GLIBCXX_BITS_SF_LEGENDRE_TCC, [619](#)
- sf_mod_bessel.tcc
 - _GLIBCXX_BITS_SF_MOD_BESSEL_TCC, [621](#)

- sf_owens.tcc
 - _GLIBCXX_BITS_SF_OWENS_T_TCC, [622](#)
- sf_polylog.tcc
 - _GLIBCXX_BITS_SF_POLYLOG_TCC, [625](#)
- sf_stirling.tcc
 - _GLIBCXX_BITS_SF_STIRLING_TCC, [627](#)
- sf_theta.tcc
 - _GLIBCXX_BITS_SF_THETA_TCC, [630](#)
- sf_trig.tcc
 - _GLIBCXX_BITS_SF_TRIG_TCC, [632](#)
- sf_trigint.tcc
 - _GLIBCXX_BITS_SF_TRIGINT_TCC, [634](#)
- sf_zeta.tcc
 - _GLIBCXX_BITS_SF_ZETA_TCC, [637](#)
- sin_pi
 - GNU Extended Mathematical Special Functions, [173](#)
- sin_pif
 - GNU Extended Mathematical Special Functions, [174](#)
- sin_pil
 - GNU Extended Mathematical Special Functions, [174](#)
- sinc
 - GNU Extended Mathematical Special Functions, [174](#)
- sinc_pi
 - GNU Extended Mathematical Special Functions, [175](#)
- sinc_pif
 - GNU Extended Mathematical Special Functions, [175](#)
- sinc_pil
 - GNU Extended Mathematical Special Functions, [176](#)
- sincf
 - GNU Extended Mathematical Special Functions, [176](#)
- sincl
 - GNU Extended Mathematical Special Functions, [176](#)
- sincos
 - GNU Extended Mathematical Special Functions, [177](#)
- sincos_pi
 - GNU Extended Mathematical Special Functions, [177](#)
- sincos_pif
 - GNU Extended Mathematical Special Functions, [178](#)
- sincos_pil
 - GNU Extended Mathematical Special Functions, [178](#)
- sincosf
 - GNU Extended Mathematical Special Functions, [178](#)
- sincosl
 - GNU Extended Mathematical Special Functions, [179](#)
- sinh_pi
 - GNU Extended Mathematical Special Functions, [179](#)
- sinh_pif
 - GNU Extended Mathematical Special Functions, [180](#)
- sinh_pil
 - GNU Extended Mathematical Special Functions, [180](#)
- sinhc
 - GNU Extended Mathematical Special Functions, [180](#)
- sinhc_pi
 - GNU Extended Mathematical Special Functions, [181](#)
- sinhc_pif
 - GNU Extended Mathematical Special Functions, [181](#)
- sinhc_pil
 - GNU Extended Mathematical Special Functions, [182](#)
- sinhcf
 - GNU Extended Mathematical Special Functions, [182](#)
- sinhcl
 - GNU Extended Mathematical Special Functions, [182](#)
- sinhint
 - GNU Extended Mathematical Special Functions, [182](#)
- sinhintf
 - GNU Extended Mathematical Special Functions, [183](#)
- sinhintl
 - GNU Extended Mathematical Special Functions, [183](#)
- sinint
 - GNU Extended Mathematical Special Functions, [183](#)
- sinintf
 - GNU Extended Mathematical Special Functions, [184](#)
- sinintl
 - GNU Extended Mathematical Special Functions, [184](#)
- specfun.h
 - __STDCPP_MATH_SPEC_FUNCS__, [653](#)
 - __cpp_lib_math_special_functions, [652](#)
- sph_bessel
 - C++17/IS29124 Mathematical Special Functions, [47](#)
- sph_bessel_i
 - GNU Extended Mathematical Special Functions, [184](#)
- sph_bessel_if
 - GNU Extended Mathematical Special Functions, [185](#)
- sph_bessel_il
 - GNU Extended Mathematical Special Functions, [185](#)
- sph_bessel_k
 - GNU Extended Mathematical Special Functions, [186](#)
- sph_bessel_kf
 - GNU Extended Mathematical Special Functions, [187](#)
- sph_bessel_kl
 - GNU Extended Mathematical Special Functions, [187](#)
- sph_besself
 - C++17/IS29124 Mathematical Special Functions, [47](#)
- sph_bessell
 - C++17/IS29124 Mathematical Special Functions, [48](#)
- sph_hankel_1
 - GNU Extended Mathematical Special Functions, [187](#), [188](#)
- sph_hankel_1f
 - GNU Extended Mathematical Special Functions, [188](#), [189](#)
- sph_hankel_1l
 - GNU Extended Mathematical Special Functions, [189](#)
- sph_hankel_2
 - GNU Extended Mathematical Special Functions, [190](#), [191](#)
- sph_hankel_2f
 - GNU Extended Mathematical Special Functions, [191](#)

- sph_hankel_2l
 - GNU Extended Mathematical Special Functions, [192](#)
- sph_harmonic
 - GNU Extended Mathematical Special Functions, [192](#)
- sph_harmonicf
 - GNU Extended Mathematical Special Functions, [193](#)
- sph_harmonicl
 - GNU Extended Mathematical Special Functions, [193](#)
- sph_legendre
 - C++17/IS29124 Mathematical Special Functions, [48](#)
- sph_legendref
 - C++17/IS29124 Mathematical Special Functions, [49](#)
- sph_legendrel
 - C++17/IS29124 Mathematical Special Functions, [49](#)
- sph_neumann
 - C++17/IS29124 Mathematical Special Functions, [49](#)
- sph_neumannf
 - C++17/IS29124 Mathematical Special Functions, [50](#)
- sph_neumannl
 - C++17/IS29124 Mathematical Special Functions, [50](#)
- std, [236](#)
- std::__detail, [238](#)
 - _Num_Euler_Maclaurin_zeta, [440](#)
 - _Num_Stieltjes, [441](#)
 - _S_Euler_Maclaurin_zeta, [441](#)
 - _S_Stieltjes, [445](#)
 - _S_double_factorial_table, [441](#)
 - _S_factorial_table, [441](#)
 - _S_harmonic_denom, [442](#)
 - _S_harmonic_numer, [442](#)
 - _S_neg_double_factorial_table, [442](#)
 - _S_num_double_factorials, [442](#)
 - _S_num_double_factorials< double >, [442](#)
 - _S_num_double_factorials< float >, [443](#)
 - _S_num_double_factorials< long double >, [443](#)
 - _S_num_factorials, [443](#)
 - _S_num_factorials< double >, [443](#)
 - _S_num_factorials< float >, [443](#)
 - _S_num_factorials< long double >, [444](#)
 - _S_num_harmonic_numer, [444](#)
 - _S_num_neg_double_factorials, [444](#)
 - _S_num_neg_double_factorials< double >, [444](#)
 - _S_num_neg_double_factorials< float >, [444](#)
 - _S_num_neg_double_factorials< long double >, [445](#)
 - _S_num_zetam1, [445](#)
 - _S_zetam1, [445](#)
 - _airy, [262](#)
 - _airy_ai, [262](#)
 - _airy_arg, [262](#)
 - _airy_bi, [263](#)
 - _assoc_laguerre, [263](#)
 - _assoc_legendre_p, [264](#)
 - _bernoulli, [265](#)
 - _bernoulli_2n, [265](#)
 - _bernoulli_series, [266](#)
 - _beta, [267](#)
 - _beta_gamma, [267](#)
 - _beta_inc, [268](#)
 - _beta_lgamma, [269](#)
 - _beta_p, [269](#)
 - _beta_product, [269](#)
 - _binomial, [270](#), [271](#)
 - _binomial_p, [271](#)
 - _binomial_pdf, [272](#)
 - _binomial_q, [273](#)
 - _bose_einstein, [273](#)
 - _cauchy_p, [274](#)
 - _chebyshev_recur, [274](#)
 - _chebyshev_t, [275](#)
 - _chebyshev_u, [275](#)
 - _chebyshev_v, [276](#)
 - _chebyshev_w, [277](#)
 - _chi_squared_pdf, [277](#)
 - _chi_squared_pdfc, [278](#)
 - _chshint, [278](#)
 - _chshint_cont_frac, [279](#)
 - _chshint_series, [279](#)
 - _clamp_0_m2pi, [279](#)
 - _clamp_pi, [280](#)
 - _clausen, [280](#), [281](#)
 - _clausen_cl, [281](#), [282](#)
 - _clausen_sl, [282](#), [283](#)
 - _comp_ellint_1, [284](#)
 - _comp_ellint_2, [284](#)
 - _comp_ellint_3, [286](#)
 - _comp_ellint_d, [286](#)
 - _comp_ellint_rf, [287](#)
 - _comp_ellint_rg, [287](#)
 - _conf_hyperg, [287](#)
 - _conf_hyperg_lim, [288](#)
 - _conf_hyperg_lim_series, [288](#)
 - _conf_hyperg_luke, [289](#)
 - _conf_hyperg_series, [289](#)
 - _cos_pi, [290](#)
 - _cosh_pi, [291](#)
 - _coshint, [291](#)
 - _coulomb_CF1, [292](#)
 - _coulomb_CF2, [292](#)
 - _coulomb_f_recur, [292](#)
 - _coulomb_g_recur, [293](#)
 - _coulomb_norm, [293](#)
 - _cyl_bessel, [293](#)
 - _cyl_bessel_i, [294](#)
 - _cyl_bessel_ij_series, [295](#)
 - _cyl_bessel_ik, [295](#)
 - _cyl_bessel_ik_asymp, [296](#)
 - _cyl_bessel_ik_steep, [297](#)
 - _cyl_bessel_j, [297](#)

- [__cyl_bessel_jn](#), 298
- [__cyl_bessel_jn_asymp](#), 298
- [__cyl_bessel_jn_neg_arg](#), 299
- [__cyl_bessel_jn_steel](#), 299
- [__cyl_bessel_k](#), 300
- [__cyl_hankel_1](#), 300, 301
- [__cyl_hankel_2](#), 302
- [__cyl_neumann](#), 303
- [__cyl_neumann_n](#), 303
- [dawson](#), 304
- [dawson_cont_frac](#), 304
- [dawson_series](#), 305
- [debye](#), 305
- [debye_region](#), 306
- [digamma](#), 306
- [digamma_asymp](#), 307
- [digamma_series](#), 307
- [dilog](#), 308
- [dirichlet_beta](#), 308, 309
- [dirichlet_eta](#), 309, 310
- [dirichlet_lambda](#), 311
- [double_factorial](#), 311
- [ellint_1](#), 312
- [ellint_2](#), 312
- [ellint_3](#), 313
- [ellint_cel](#), 314
- [ellint_d](#), 314
- [ellint_el1](#), 314
- [ellint_el2](#), 314
- [ellint_el3](#), 315
- [ellint_rc](#), 315
- [ellint_rd](#), 316
- [ellint_rf](#), 317
- [ellint_rg](#), 317
- [ellint_rj](#), 318
- [ellnome](#), 319
- [ellnome_k](#), 320
- [ellnome_series](#), 320
- [euler](#), 320, 321
- [euler_series](#), 321
- [eulerian_1](#), 321
- [eulerian_1_recur](#), 322
- [eulerian_2](#), 322
- [eulerian_2_recur](#), 322
- [exp2](#), 323
- [expint](#), 323, 324
- [expint_E1](#), 324
- [expint_E1_asymp](#), 325
- [expint_E1_series](#), 326
- [expint_Ei](#), 326
- [expint_Ei_asymp](#), 327
- [expint_Ei_series](#), 327
- [expint_En_asymp](#), 328
- [expint_En_cont_frac](#), 329
- [expint_En_large_n](#), 329
- [expint_En_recursion](#), 330
- [expint_En_series](#), 330
- [exponential_p](#), 331
- [exponential_pdf](#), 331
- [exponential_q](#), 332
- [factorial](#), 332
- [falling_factorial](#), 332, 333
- [fermi_dirac](#), 333
- [fisher_f_p](#), 334
- [fisher_f_pdf](#), 334
- [fisher_f_q](#), 335
- [fock_airy](#), 336
- [fresnel](#), 336
- [fresnel_cont_frac](#), 337
- [fresnel_series](#), 337
- [gamma](#), 337, 338
- [gamma_cont_frac](#), 338
- [gamma_p](#), 338, 339
- [gamma_pdf](#), 339
- [gamma_q](#), 340
- [gamma_reciprocal](#), 341
- [gamma_reciprocal_series](#), 341
- [gamma_series](#), 342
- [gamma_temme](#), 342
- [gauss](#), 343
- [gegenbauer_recur](#), 343
- [gegenbauer_zeros](#), 344
- [hankel](#), 344
- [hankel_debye](#), 345
- [hankel_params](#), 345
- [hankel_uniform](#), 346
- [hankel_uniform_olver](#), 347
- [hankel_uniform_outer](#), 347
- [hankel_uniform_sum](#), 348
- [harmonic_number](#), 349
- [hermite](#), 349
- [hermite_asymp](#), 350
- [hermite_recur](#), 351
- [hermite_zeros](#), 352
- [heuman_lambda](#), 352
- [hurwitz_zeta](#), 352
- [hurwitz_zeta_euler_maclaurin](#), 353
- [hurwitz_zeta_polylog](#), 353
- [hydrogen](#), 354
- [hyperg](#), 354
- [hyperg_luke](#), 355
- [hyperg_recur](#), 355
- [hyperg_reflect](#), 356
- [hyperg_series](#), 357
- [ibeta_cont_frac](#), 358
- [jacobi_ellint](#), 358
- [jacobi_recur](#), 359
- [jacobi_theta_1](#), 359, 360

- __jacobi_theta_1_prod, 361
- __jacobi_theta_1_sum, 361
- __jacobi_theta_2, 362, 363
- __jacobi_theta_2_prod, 363
- __jacobi_theta_2_sum, 364
- __jacobi_theta_3, 365
- __jacobi_theta_3_prod, 366
- __jacobi_theta_3_sum, 367
- __jacobi_theta_4, 367, 368
- __jacobi_theta_4_prod, 369
- __jacobi_theta_4_sum, 369
- __jacobi_zeros, 370
- __jacobi_zeta, 370
- __kolmogorov_p, 371
- __laguerre, 371, 372
- __laguerre_hyperg, 372
- __laguerre_large_n, 373
- __laguerre_recur, 374
- __laguerre_zeros, 375
- __lanczos_binet1p, 375
- __lanczos_log_gamma1p, 376
- __legendre_p, 376
- __legendre_q, 377
- __legendre_zeros, 378
- __log_binomial, 378, 379
- __log_binomial_sign, 379, 380
- __log_double_factorial, 380
- __log_factorial, 381
- __log_falling_factorial, 381
- __log_gamma, 382, 383
- __log_gamma_bernoulli, 383
- __log_gamma_sign, 384
- __log_rising_factorial, 384
- __log_stirling_1, 385
- __log_stirling_1_sign, 385
- __log_stirling_2, 385
- __logint, 386
- __logistic_p, 386
- __logistic_pdf, 387
- __lognormal_p, 387
- __lognormal_pdf, 387
- __max_FGH, 440
- __max_FGH< double >, 440
- __max_FGH< float >, 440
- __normal_p, 388
- __normal_pdf, 388
- __owens_t, 388
- __polar_pi, 389
- __polygamma, 390
- __polylog, 390, 391
- __polylog_exp, 391
- __polylog_exp_asymp, 392
- __polylog_exp_neg, 393, 394
- __polylog_exp_neg_int, 394, 395
- __polylog_exp_neg_real, 395, 396
- __polylog_exp_pos, 397, 398
- __polylog_exp_pos_int, 399
- __polylog_exp_pos_real, 400, 401
- __polylog_exp_sum, 401
- __prob_hermite_recur, 402
- __radial_jacobi, 403
- __radial_jacobi_zeros, 404
- __rice_pdf, 404
- __riemann_zeta, 405
- __riemann_zeta_euler_maclaurin, 405
- __riemann_zeta_glob, 406
- __riemann_zeta_laurent, 406
- __riemann_zeta_m_1, 406
- __riemann_zeta_m_1_glob, 407
- __riemann_zeta_product, 407
- __riemann_zeta_sum, 408
- __rising_factorial, 408, 409
- __sin_pi, 409, 410
- __sinc, 410
- __sinc_pi, 410
- __sincos, 411
- __sincos_pi, 412
- __sincosint, 412
- __sincosint_asymp, 412
- __sincosint_cont_frac, 413
- __sincosint_series, 413
- __sinh_pi, 413, 414
- __sinhc, 414
- __sinhc_pi, 414
- __sinhint, 415
- __sph_bessel, 415, 416
- __sph_bessel_ik, 417
- __sph_bessel_jn, 417
- __sph_bessel_jn_neg_arg, 418
- __sph_hankel, 418
- __sph_hankel_1, 419
- __sph_hankel_2, 420, 421
- __sph_harmonic, 421
- __sph_legendre, 422
- __sph_neumann, 423
- __spouge_binet1p, 424
- __spouge_log_gamma1p, 424
- __stirling_1, 425
- __stirling_1_recur, 426
- __stirling_1_series, 426
- __stirling_2, 427
- __stirling_2_recur, 427
- __stirling_2_series, 427
- __student_t_p, 428
- __student_t_pdf, 428
- __student_t_q, 429
- __tan_pi, 429, 430
- __tanh_pi, 430

- [__tgamma, 431](#)
- [__tgamma_lower, 431](#)
- [__theta_1, 431](#)
- [__theta_2, 432](#)
- [__theta_2_asymp, 433](#)
- [__theta_2_sum, 433](#)
- [__theta_3, 433](#)
- [__theta_3_asymp, 434](#)
- [__theta_3_sum, 434](#)
- [__theta_4, 434](#)
- [__theta_c, 435](#)
- [__theta_d, 435](#)
- [__theta_n, 436](#)
- [__theta_s, 436](#)
- [__tricoli_u, 436](#)
- [__tricoli_u_naive, 437](#)
- [__weibull_p, 438](#)
- [__weibull_pdf, 438](#)
- [__zernike, 438](#)
- [__znorm1, 439](#)
- [__znorm2, 440](#)
- [std::__detail::__gamma_lanczos_data< _Tp >, 505](#)
- [std::__detail::__gamma_lanczos_data< double >, 505](#)
- [_S_cheby, 506](#)
- [_S_g, 506](#)
- [std::__detail::__gamma_lanczos_data< float >, 506](#)
- [_S_cheby, 507](#)
- [_S_g, 507](#)
- [std::__detail::__gamma_lanczos_data< long double >, 507](#)
- [_S_cheby, 508](#)
- [_S_g, 508](#)
- [std::__detail::__gamma_spouge_data< _Tp >, 509](#)
- [std::__detail::__gamma_spouge_data< double >, 509](#)
- [_S_cheby, 509](#)
- [std::__detail::__gamma_spouge_data< float >, 510](#)
- [_S_cheby, 510](#)
- [std::__detail::__gamma_spouge_data< long double >, 511](#)
- [_S_cheby, 511](#)
- [std::__detail::__jacobi_lattice_t](#)
- [_Cmplx, 514](#)
- [_M_omega_1, 518](#)
- [_M_omega_3, 518](#)
- [_Real, 514](#)
- [_Real_Omega1, 514](#)
- [_Real_Omega3, 514](#)
- [_S_pi, 518](#)
- [_Tp_Nome, 515](#)
- [_ellnome, 516](#)
- [_jacobi_lattice_t, 515](#)
- [_omega_1, 516](#)
- [_omega_2, 516](#)
- [_omega_3, 517](#)
- [_reduce, 517](#)
- [_tau, 518](#)
- [std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >, 512](#)
- [std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__arg_t, 519](#)
- [std::__detail::__jacobi_lattice_t< _Tp_Omega1, _Tp_Omega3 >::__tau_t, 520](#)
- [std::__detail::__jacobi_lattice_t::__arg_t](#)
- [_m, 519](#)
- [_n, 520](#)
- [_z, 520](#)
- [std::__detail::__jacobi_lattice_t::__tau_t](#)
- [_tau_t, 521](#)
- [_val, 521](#)
- [std::__detail::__jacobi_theta_0_t](#)
- [_Cmplx, 523](#)
- [_Real, 523](#)
- [_Type, 523](#)
- [_jacobi_theta_0_t, 523](#)
- [dedekind_eta, 524](#)
- [eta_1, 524](#)
- [eta_2, 524](#)
- [eta_3, 525](#)
- [th1p, 525](#)
- [th1ppp, 525](#)
- [th2, 525](#)
- [th2pp, 525](#)
- [th3, 526](#)
- [th3pp, 526](#)
- [th4, 526](#)
- [th4pp, 526](#)
- [std::__detail::__jacobi_theta_0_t< _Tp1, _Tp3 >, 522](#)
- [std::__detail::__weierstrass_invariants_t](#)
- [_Cmplx, 528](#)
- [_Real, 528](#)
- [_Type, 528](#)
- [_delta, 529](#)
- [_g_2, 529](#)
- [_g_3, 529](#)
- [_klein_j, 529](#)
- [_weierstrass_invariants_t, 528](#)
- [std::__detail::__weierstrass_invariants_t< _Tp1, _Tp3 >, 527](#)
- [std::__detail::__weierstrass_roots_t](#)
- [_Cmplx, 531](#)
- [_Real, 531](#)
- [_Type, 531](#)
- [_delta, 532](#)
- [_e1, 533](#)
- [_e2, 533](#)
- [_e3, 533](#)
- [_weierstrass_roots_t, 532](#)
- [std::__detail::__weierstrass_roots_t< _Tp1, _Tp3 >, 530](#)

std::__detail::_Airy
 _Airy, 535
 inner_radius, 536
 operator(), 535
 outer_radius, 536
 scalar_type, 534
 value_type, 534
 std::__detail::_Airy<_Tp>, 534
 std::__detail::_Airy_asyp
 _Airy_asyp, 538
 _Cmplx, 538
 _S_absarg_ge_pio3, 538
 _S_absarg_lt_pio3, 539
 operator(), 539
 std::__detail::_Airy_asyp<_Tp>, 536
 std::__detail::_Airy_asyp_data<_Tp>, 540
 std::__detail::_Airy_asyp_data<double>, 541
 _S_c, 541
 _S_d, 541
 _S_max_cd, 541
 std::__detail::_Airy_asyp_data<float>, 542
 _S_c, 542
 _S_d, 542
 _S_max_cd, 542
 std::__detail::_Airy_asyp_data<long double>, 543
 _S_c, 543
 _S_d, 543
 _S_max_cd, 543
 std::__detail::_Airy_asyp_series
 _Airy_asyp_series, 545
 _S_sqrt_pi, 546
 operator(), 546
 scalar_type, 545
 value_type, 545
 std::__detail::_Airy_asyp_series<_Sum>, 544
 std::__detail::_Airy_default_radII<_Tp>, 547
 std::__detail::_Airy_default_radII<double>, 547
 inner_radius, 547
 outer_radius, 547
 std::__detail::_Airy_default_radII<float>, 548
 inner_radius, 548
 outer_radius, 548
 std::__detail::_Airy_default_radII<long double>, 549
 inner_radius, 549
 outer_radius, 549
 std::__detail::_Airy_series
 _Cmplx, 550
 _N_FGH, 555
 _S_Ai, 551
 _S_Ai0, 555
 _S_Aip0, 555
 _S_Airy, 551
 _S_Bi, 552
 _S_Bi0, 555
 _S_Bip0, 556
 _S_FGH, 552
 _S_Fock, 553
 _S_Gi0, 556
 _S_Gip0, 556
 _S_Hi0, 556
 _S_Hip0, 557
 _S_Scorer, 553
 _S_Scorer2, 554
 _S_eps, 556
 _S_i, 557
 _S_pi, 557
 _S_sqrt_pi, 557
 std::__detail::_Airy_series<_Tp>, 549
 std::__detail::_AiryAuxilliaryState
 _Val, 558
 __fai_deriv, 558
 __fai_value, 559
 __gai_deriv, 559
 __gai_value, 559
 __hai_deriv, 559
 __hai_value, 559
 __z, 560
 std::__detail::_AiryAuxilliaryState<_Tp>, 558
 std::__detail::_AiryState
 _Real, 561
 __Ai_deriv, 562
 __Ai_value, 562
 __Bi_deriv, 562
 __Bi_value, 562
 __z, 562
 true_Wronskian, 561
 Wronskian, 561
 std::__detail::_AiryState<_Tp>, 560
 std::__detail::_AsympTerminator
 _AsympTerminator, 563
 num_terms, 564
 operator<<, 564
 operator(), 564
 std::__detail::_AsympTerminator<_Tp>, 563
 std::__detail::_Factorial_table
 __factorial, 565
 __log_factorial, 565
 __n, 565
 std::__detail::_Factorial_table<_Tp>, 565
 std::__detail::_Terminator
 _Terminator, 566
 num_terms, 567
 operator(), 567
 std::__detail::_Terminator<_Tp>, 566
 stirling_1
 GNU Extended Mathematical Special Functions, 194
 stirling_2
 GNU Extended Mathematical Special Functions, 194

- student_t_p
 - GNU Extended Mathematical Special Functions, [195](#)
- student_t_pdf
 - GNU Extended Mathematical Special Functions, [195](#)
- tan_pi
 - GNU Extended Mathematical Special Functions, [196](#)
- tan_pif
 - GNU Extended Mathematical Special Functions, [196](#)
- tan_pil
 - GNU Extended Mathematical Special Functions, [197](#)
- tanh_pi
 - GNU Extended Mathematical Special Functions, [197](#)
- tanh_pif
 - GNU Extended Mathematical Special Functions, [198](#)
- tanh_pil
 - GNU Extended Mathematical Special Functions, [198](#)
- tgamma
 - GNU Extended Mathematical Special Functions, [198](#), [199](#)
- tgamma_lower
 - GNU Extended Mathematical Special Functions, [199](#)
- tgamma_lowerf
 - GNU Extended Mathematical Special Functions, [199](#)
- tgamma_lowerl
 - GNU Extended Mathematical Special Functions, [200](#)
- tgammaf
 - GNU Extended Mathematical Special Functions, [200](#), [201](#)
- tgammal
 - GNU Extended Mathematical Special Functions, [201](#), [202](#)
- th1p
 - std::__detail::__jacobi_theta_0_t, [525](#)
- th1ppp
 - std::__detail::__jacobi_theta_0_t, [525](#)
- th2
 - std::__detail::__jacobi_theta_0_t, [525](#)
- th2pp
 - std::__detail::__jacobi_theta_0_t, [525](#)
- th3
 - std::__detail::__jacobi_theta_0_t, [526](#)
- th3pp
 - std::__detail::__jacobi_theta_0_t, [526](#)
- th4
 - std::__detail::__jacobi_theta_0_t, [526](#)
- th4pp
 - std::__detail::__jacobi_theta_0_t, [526](#)
- theta_1
 - GNU Extended Mathematical Special Functions, [202](#)
- theta_1f
 - GNU Extended Mathematical Special Functions, [203](#)
- theta_1l
 - GNU Extended Mathematical Special Functions, [203](#)
- theta_2
 - GNU Extended Mathematical Special Functions, [203](#)
- theta_2f
 - GNU Extended Mathematical Special Functions, [204](#)
- theta_2l
 - GNU Extended Mathematical Special Functions, [204](#)
- theta_3
 - GNU Extended Mathematical Special Functions, [204](#)
- theta_3f
 - GNU Extended Mathematical Special Functions, [205](#)
- theta_3l
 - GNU Extended Mathematical Special Functions, [205](#)
- theta_4
 - GNU Extended Mathematical Special Functions, [205](#)
- theta_4f
 - GNU Extended Mathematical Special Functions, [206](#)
- theta_4l
 - GNU Extended Mathematical Special Functions, [206](#)
- theta_c
 - GNU Extended Mathematical Special Functions, [206](#)
- theta_cf
 - GNU Extended Mathematical Special Functions, [207](#)
- theta_cl
 - GNU Extended Mathematical Special Functions, [207](#)
- theta_d
 - GNU Extended Mathematical Special Functions, [208](#)
- theta_df
 - GNU Extended Mathematical Special Functions, [208](#)
- theta_dl
 - GNU Extended Mathematical Special Functions, [209](#)
- theta_n
 - GNU Extended Mathematical Special Functions, [209](#)
- theta_nf
 - GNU Extended Mathematical Special Functions, [210](#)
- theta_nl
 - GNU Extended Mathematical Special Functions, [210](#)
- theta_s
 - GNU Extended Mathematical Special Functions, [210](#)
- theta_sf
 - GNU Extended Mathematical Special Functions, [211](#)
- theta_sl
 - GNU Extended Mathematical Special Functions, [211](#)
- tricomi_u
 - GNU Extended Mathematical Special Functions, [212](#)
- tricomi_uf
 - GNU Extended Mathematical Special Functions, [213](#)
- tricomi_ul
 - GNU Extended Mathematical Special Functions, [213](#)
- true_Wronskian
 - std::__detail::__AiryState, [561](#)
- value_type
 - std::__detail::__Airy, [534](#)
 - std::__detail::__Airy_asymp_series, [545](#)

weibull_p

GNU Extended Mathematical Special Functions, [213](#)

weibull_pdf

GNU Extended Mathematical Special Functions, [213](#)

Wronskian

std::__detail::_AiryState, [561](#)

zernike

GNU Extended Mathematical Special Functions, [214](#)

zernikef

GNU Extended Mathematical Special Functions, [215](#)

zernikel

GNU Extended Mathematical Special Functions, [215](#)