

TR29124 C++ Special Math Functions

2.0

Generated by Doxygen 1.8.9.1

Thu Apr 28 2016 16:53:19

Contents

1	Mathematical Special Functions	1
1.1	Introduction and History	1
1.2	Contents	1
1.3	General Features	3
1.3.1	Argument Promotion	3
1.3.2	NaN Arguments	3
1.4	Implementation	3
1.5	Testing	3
1.6	General Bibliography	4
2	Todo List	5
3	Module Index	7
3.1	Modules	7
4	Namespace Index	9
4.1	Namespace List	9
5	Class Index	11
5.1	Class List	11
6	File Index	13
6.1	File List	13
7	Module Documentation	15
7.1	C++ Mathematical Special Functions	15
7.1.1	Detailed Description	15
7.2	C++17/IS29124 Mathematical Special Functions	16
7.2.1	Detailed Description	18
7.2.2	Function Documentation	18
7.2.2.1	assoc_laguerre	18

7.2.2.2	assoc_laguerref	19
7.2.2.3	assoc_laguerrel	19
7.2.2.4	assoc_legendre	19
7.2.2.5	assoc_legendref	20
7.2.2.6	assoc_legendrel	20
7.2.2.7	beta	20
7.2.2.8	betaf	20
7.2.2.9	betal	21
7.2.2.10	comp_ellint_1	21
7.2.2.11	comp_ellint_1f	21
7.2.2.12	comp_ellint_1l	22
7.2.2.13	comp_ellint_2	22
7.2.2.14	comp_ellint_2f	22
7.2.2.15	comp_ellint_2l	22
7.2.2.16	comp_ellint_3	23
7.2.2.17	comp_ellint_3f	23
7.2.2.18	comp_ellint_3l	23
7.2.2.19	cyl_bessel_i	24
7.2.2.20	cyl_bessel_if	24
7.2.2.21	cyl_bessel_il	24
7.2.2.22	cyl_bessel_j	24
7.2.2.23	cyl_bessel_jf	25
7.2.2.24	cyl_bessel_jl	25
7.2.2.25	cyl_bessel_k	25
7.2.2.26	cyl_bessel_kf	26
7.2.2.27	cyl_bessel_kl	26
7.2.2.28	cyl_neumann	26
7.2.2.29	cyl_neumannf	27
7.2.2.30	cyl_neumannl	27
7.2.2.31	ellint_1	27
7.2.2.32	ellint_1f	28
7.2.2.33	ellint_1l	28
7.2.2.34	ellint_2	28
7.2.2.35	ellint_2f	29
7.2.2.36	ellint_2l	29
7.2.2.37	ellint_3	29
7.2.2.38	ellint_3f	30

7.2.2.39	ellint_3l	30
7.2.2.40	expint	30
7.2.2.41	expintf	31
7.2.2.42	expintl	31
7.2.2.43	hermite	31
7.2.2.44	hermitef	32
7.2.2.45	hermitel	32
7.2.2.46	laguerre	32
7.2.2.47	laguerref	32
7.2.2.48	laguerrel	33
7.2.2.49	legendre	33
7.2.2.50	legendref	33
7.2.2.51	legendrel	33
7.2.2.52	riemann_zeta	34
7.2.2.53	riemann_zetaf	34
7.2.2.54	riemann_zetal	34
7.2.2.55	sph_bessel	35
7.2.2.56	sph_besself	35
7.2.2.57	sph_bessell	35
7.2.2.58	sph_legendre	35
7.2.2.59	sph_legendref	36
7.2.2.60	sph_legendrel	36
7.2.2.61	sph_neumann	36
7.2.2.62	sph_neumannf	37
7.2.2.63	sph_neumannl	37
7.3	GNU Extended Mathematical Special Functions	38
7.3.1	Detailed Description	46
7.3.2	Enumeration Type Documentation	46
7.3.2.1	anonymous enum	46
7.3.3	Function Documentation	46
7.3.3.1	airy_ai	46
7.3.3.2	airy_aif	47
7.3.3.3	airy_ail	47
7.3.3.4	airy_bi	47
7.3.3.5	airy_bif	47
7.3.3.6	airy_bil	48
7.3.3.7	bernoulli	48

7.3.3.8	bernoullif	48
7.3.3.9	bernoullil	48
7.3.3.10	bincoef	48
7.3.3.11	bincoeff	48
7.3.3.12	bincoefl	49
7.3.3.13	chebyshev_t	49
7.3.3.14	chebyshev_tf	49
7.3.3.15	chebyshev_tl	49
7.3.3.16	chebyshev_u	49
7.3.3.17	chebyshev_uf	50
7.3.3.18	chebyshev_ul	50
7.3.3.19	chebyshev_v	50
7.3.3.20	chebyshev_vf	51
7.3.3.21	chebyshev_vl	51
7.3.3.22	chebyshev_w	51
7.3.3.23	chebyshev_wf	51
7.3.3.24	chebyshev_wl	52
7.3.3.25	clausen	52
7.3.3.26	clausen	52
7.3.3.27	clausen_c	52
7.3.3.28	clausen_cf	52
7.3.3.29	clausen_cl	53
7.3.3.30	clausen_s	53
7.3.3.31	clausen_sf	53
7.3.3.32	clausen_sl	53
7.3.3.33	clausenf	53
7.3.3.34	clausenf	54
7.3.3.35	clausenl	54
7.3.3.36	clausenl	54
7.3.3.37	comp_ellint_d	54
7.3.3.38	comp_ellint_df	54
7.3.3.39	comp_ellint_dl	54
7.3.3.40	comp_ellint_rf	55
7.3.3.41	comp_ellint_rf	55
7.3.3.42	comp_ellint_rf	55
7.3.3.43	comp_ellint_rg	55
7.3.3.44	comp_ellint_rg	56

7.3.3.45	comp_ellint_rg	56
7.3.3.46	conf_hyperg	56
7.3.3.47	conf_hyperg_lim	56
7.3.3.48	conf_hyperg_limf	58
7.3.3.49	conf_hyperg_liml	58
7.3.3.50	conf_hypergf	58
7.3.3.51	conf_hypergl	58
7.3.3.52	coshint	59
7.3.3.53	coshintf	59
7.3.3.54	coshintl	59
7.3.3.55	cosint	59
7.3.3.56	cosintf	60
7.3.3.57	cosintl	60
7.3.3.58	cyl_hankel_1	60
7.3.3.59	cyl_hankel_1	61
7.3.3.60	cyl_hankel_1f	61
7.3.3.61	cyl_hankel_1f	61
7.3.3.62	cyl_hankel_1l	61
7.3.3.63	cyl_hankel_1l	62
7.3.3.64	cyl_hankel_2	62
7.3.3.65	cyl_hankel_2	62
7.3.3.66	cyl_hankel_2f	63
7.3.3.67	cyl_hankel_2f	63
7.3.3.68	cyl_hankel_2l	63
7.3.3.69	cyl_hankel_2l	63
7.3.3.70	dawson	64
7.3.3.71	dawsonf	64
7.3.3.72	dawsonl	64
7.3.3.73	digamma	64
7.3.3.74	digammaf	64
7.3.3.75	digammal	64
7.3.3.76	dilog	65
7.3.3.77	dilogf	65
7.3.3.78	dilogl	65
7.3.3.79	dirichlet_beta	65
7.3.3.80	dirichlet_betaf	66
7.3.3.81	dirichlet_betall	66

7.3.3.82	dirichlet_eta	66
7.3.3.83	dirichlet_etaf	66
7.3.3.84	dirichlet_etat	67
7.3.3.85	double_factorial	67
7.3.3.86	double_factorialf	67
7.3.3.87	double_factoriall	67
7.3.3.88	ellint_cel	67
7.3.3.89	ellint_celf	67
7.3.3.90	ellint_cell	68
7.3.3.91	ellint_d	68
7.3.3.92	ellint_df	68
7.3.3.93	ellint_dl	68
7.3.3.94	ellint_el1	68
7.3.3.95	ellint_el1f	69
7.3.3.96	ellint_el1l	69
7.3.3.97	ellint_el2	69
7.3.3.98	ellint_el2f	70
7.3.3.99	ellint_el2l	70
7.3.3.100	ellint_el3	70
7.3.3.101	ellint_el3f	70
7.3.3.102	ellint_el3l	71
7.3.3.103	ellint_rc	71
7.3.3.104	ellint_rcf	71
7.3.3.105	ellint_rcl	71
7.3.3.106	ellint_rd	72
7.3.3.107	ellint_rdf	72
7.3.3.108	ellint_rdl	72
7.3.3.109	ellint_rf	72
7.3.3.110	ellint_rff	73
7.3.3.111	ellint_rfl	73
7.3.3.112	ellint_rg	73
7.3.3.113	ellint_rgf	74
7.3.3.114	ellint_rgl	74
7.3.3.115	ellint_rj	74
7.3.3.116	ellint_rjf	75
7.3.3.117	ellint_rjl	75
7.3.3.118	ellnome	75

7.3.3.119 ellnomef	75
7.3.3.120 ellnomel	75
7.3.3.121 expint_e1	76
7.3.3.122 expint_e1f	76
7.3.3.123 expint_e1l	76
7.3.3.124 expint_en	76
7.3.3.125 expint_enf	76
7.3.3.126 expint_enl	76
7.3.3.127 factorial	76
7.3.3.128 factorialf	76
7.3.3.129 factoriall	76
7.3.3.130 fresnel_c	77
7.3.3.131 fresnel_cf	77
7.3.3.132 fresnel_cl	77
7.3.3.133 fresnel_s	77
7.3.3.134 fresnel_sf	77
7.3.3.135 fresnel_sl	77
7.3.3.136 gamma_l	77
7.3.3.137 gamma_lf	77
7.3.3.138 gamma_ll	78
7.3.3.139 gamma_p	78
7.3.3.140 gamma_pf	78
7.3.3.141 gamma_pl	78
7.3.3.142 gamma_q	78
7.3.3.143 gamma_qf	78
7.3.3.144 gamma_ql	78
7.3.3.145 gamma_u	78
7.3.3.146 gamma_uf	78
7.3.3.147 gamma_ul	78
7.3.3.148 gegenbauer	79
7.3.3.149 gegenbauerf	79
7.3.3.150 gegenbauerl	79
7.3.3.151 heuman_lambda	79
7.3.3.152 heuman_lambdaf	80
7.3.3.153 heuman_lambdal	80
7.3.3.154 hurwitz_zeta	80
7.3.3.155 hurwitz_zeta	80

7.3.3.156 hurwitz_zetaf	80
7.3.3.157 hurwitz_zetal	81
7.3.3.158 hyperg	81
7.3.3.159 hypergf	81
7.3.3.160 hypergl	81
7.3.3.161 ibeta	82
7.3.3.162 ibetac	82
7.3.3.163 ibetacf	82
7.3.3.164 ibetacl	82
7.3.3.165 ibetaf	83
7.3.3.166 ibetal	83
7.3.3.167 jacobi	83
7.3.3.168 jacobi_cn	83
7.3.3.169 jacobi_cnf	84
7.3.3.170 jacobi_cnl	84
7.3.3.171 jacobi_dn	84
7.3.3.172 jacobi_dnf	84
7.3.3.173 jacobi_dnl	85
7.3.3.174 jacobi_sn	85
7.3.3.175 jacobi_snf	85
7.3.3.176 jacobi_snl	85
7.3.3.177 jacobi_zeta	86
7.3.3.178 jacobi_zetaf	86
7.3.3.179 jacobi_zetal	86
7.3.3.180 jacobif	86
7.3.3.181 jacobil	86
7.3.3.182 lbincoef	87
7.3.3.183 lbincoeff	87
7.3.3.184 lbincoefl	87
7.3.3.185 ldouble_factorial	87
7.3.3.186 ldouble_factorialf	87
7.3.3.187 ldouble_factoriall	87
7.3.3.188 legendre_q	87
7.3.3.189 legendre_qf	87
7.3.3.190 legendre_ql	87
7.3.3.191 lfactorial	88
7.3.3.192 lfactorialf	88

7.3.3.193 lfactorial	88
7.3.3.194 logint	88
7.3.3.195 logintf	88
7.3.3.196 logintl	88
7.3.3.197 lpochhammer_l	88
7.3.3.198 lpochhammer_lf	89
7.3.3.199 lpochhammer_ll	89
7.3.3.200 lpochhammer_u	89
7.3.3.201 lpochhammer_uf	89
7.3.3.202 lpochhammer_ul	89
7.3.3.203 owens_t	89
7.3.3.204 owens_tf	89
7.3.3.205 owens_tl	90
7.3.3.206 pochhammer_l	90
7.3.3.207 pochhammer_lf	90
7.3.3.208 pochhammer_ll	90
7.3.3.209 pochhammer_u	90
7.3.3.210 pochhammer_uf	90
7.3.3.211 pochhammer_ul	90
7.3.3.212 polylog	90
7.3.3.213 polylog	91
7.3.3.214 polylogf	91
7.3.3.215 polylogf	91
7.3.3.216 polylogl	91
7.3.3.217 polylogl	92
7.3.3.218 psi	92
7.3.3.219 psif	92
7.3.3.220 psil	92
7.3.3.221 radpoly	92
7.3.3.222 radpolyf	93
7.3.3.223 radpolyl	93
7.3.3.224 sinc	93
7.3.3.225 sinc_pi	93
7.3.3.226 sinc_pif	93
7.3.3.227 sinc_pil	93
7.3.3.228 sincf	94
7.3.3.229 sincl	94

7.3.3.230 sinhc	94
7.3.3.231 sinhc_pi	94
7.3.3.232 sinhc_pif	94
7.3.3.233 sinhc_pil	94
7.3.3.234 sinhcf	94
7.3.3.235 sinhcl	94
7.3.3.236 sinhint	94
7.3.3.237 sinhintf	95
7.3.3.238 sinhintl	95
7.3.3.239 sinint	95
7.3.3.240 sinintf	95
7.3.3.241 sinintl	96
7.3.3.242 sph_bessel_i	96
7.3.3.243 sph_bessel_if	96
7.3.3.244 sph_bessel_il	96
7.3.3.245 sph_bessel_k	97
7.3.3.246 sph_bessel_kf	97
7.3.3.247 sph_bessel_kl	97
7.3.3.248 sph_hankel_1	97
7.3.3.249 sph_hankel_1l	98
7.3.3.250 sph_hankel_1f	98
7.3.3.251 sph_hankel_1f	98
7.3.3.252 sph_hankel_1l	98
7.3.3.253 sph_hankel_1l	99
7.3.3.254 sph_hankel_2	99
7.3.3.255 sph_hankel_2	99
7.3.3.256 sph_hankel_2f	100
7.3.3.257 sph_hankel_2f	100
7.3.3.258 sph_hankel_2l	100
7.3.3.259 sph_hankel_2l	100
7.3.3.260 sph_harmonic	101
7.3.3.261 sph_harmonicf	101
7.3.3.262 sph_harmonicl	101
7.3.3.263 theta_1	101
7.3.3.264 theta_1f	102
7.3.3.265 theta_1l	102
7.3.3.266 theta_2	102

7.3.3.267	theta_2f	102
7.3.3.268	theta_2l	103
7.3.3.269	theta_3	103
7.3.3.270	theta_3f	103
7.3.3.271	theta_3l	103
7.3.3.272	theta_4	103
7.3.3.273	theta_4f	104
7.3.3.274	theta_4l	104
7.3.3.275	theta_c	104
7.3.3.276	theta_cf	104
7.3.3.277	theta_cl	105
7.3.3.278	theta_d	105
7.3.3.279	theta_df	105
7.3.3.280	theta_dl	105
7.3.3.281	theta_n	105
7.3.3.282	theta_nf	106
7.3.3.283	theta_nl	106
7.3.3.284	theta_s	106
7.3.3.285	theta_sf	106
7.3.3.286	theta_sl	107
7.3.3.287	zernike	107
7.3.3.288	zernikef	107
7.3.3.289	zernikel	108
8	Namespace Documentation	109
8.1	__gnu_cxx Namespace Reference	109
8.2	std Namespace Reference	117
8.3	std::__detail Namespace Reference	119
8.3.1	Enumeration Type Documentation	137
8.3.1.1	anonymous enum	137
8.3.2	Function Documentation	137
8.3.2.1	__airy	137
8.3.2.2	__airy	137
8.3.2.3	__airy_ai	139
8.3.2.4	__airy_arg	139
8.3.2.5	__airy_asymp_absarg_ge_pio3	140
8.3.2.6	__airy_asymp_absarg_lt_pio3	140

8.3.2.7	__airy_bessel_i	140
8.3.2.8	__airy_bessel_k	141
8.3.2.9	__airy_bi	142
8.3.2.10	__airy_hyperg_rational	142
8.3.2.11	__assoc_laguerre	143
8.3.2.12	__assoc_legendre_p	143
8.3.2.13	__bernoulli	144
8.3.2.14	__bernoulli_2n	144
8.3.2.15	__bernoulli_series	144
8.3.2.16	__beta	145
8.3.2.17	__beta_gamma	145
8.3.2.18	__beta_inc	145
8.3.2.19	__beta_inc_cont_frac	146
8.3.2.20	__beta_lgamma	146
8.3.2.21	__beta_product	146
8.3.2.22	__bincoef	147
8.3.2.23	__binomial_cdf	147
8.3.2.24	__binomial_cdfc	148
8.3.2.25	__bose_einstein	148
8.3.2.26	__chebyshev_recur	148
8.3.2.27	__chebyshev_t	149
8.3.2.28	__chebyshev_u	149
8.3.2.29	__chebyshev_v	150
8.3.2.30	__chebyshev_w	150
8.3.2.31	__chi_squared_pdf	150
8.3.2.32	__chi_squared_pdfc	151
8.3.2.33	__chshint	151
8.3.2.34	__chshint_cont_frac	151
8.3.2.35	__chshint_series	151
8.3.2.36	__clamp_0_m2pi	151
8.3.2.37	__clamp_pi	152
8.3.2.38	__clausen	152
8.3.2.39	__clausen	152
8.3.2.40	__clausen_c	152
8.3.2.41	__clausen_c	153
8.3.2.42	__clausen_s	153
8.3.2.43	__clausen_s	154

8.3.2.44	__comp_ellint_1	154
8.3.2.45	__comp_ellint_2	154
8.3.2.46	__comp_ellint_3	155
8.3.2.47	__comp_ellint_d	155
8.3.2.48	__comp_ellint_rf	155
8.3.2.49	__comp_ellint_rg	155
8.3.2.50	__conf_hyperg	156
8.3.2.51	__conf_hyperg_lim	156
8.3.2.52	__conf_hyperg_lim_series	156
8.3.2.53	__conf_hyperg_luke	157
8.3.2.54	__conf_hyperg_series	157
8.3.2.55	__coshint	157
8.3.2.56	__cyl_bessel	157
8.3.2.57	__cyl_bessel_i	158
8.3.2.58	__cyl_bessel_ij_series	158
8.3.2.59	__cyl_bessel_ik	159
8.3.2.60	__cyl_bessel_ik_asymp	159
8.3.2.61	__cyl_bessel_ik_steel	160
8.3.2.62	__cyl_bessel_j	160
8.3.2.63	__cyl_bessel_jn	160
8.3.2.64	__cyl_bessel_jn_asymp	161
8.3.2.65	__cyl_bessel_jn_steel	161
8.3.2.66	__cyl_bessel_k	161
8.3.2.67	__cyl_hankel_1	162
8.3.2.68	__cyl_hankel_1	162
8.3.2.69	__cyl_hankel_2	162
8.3.2.70	__cyl_hankel_2	164
8.3.2.71	__cyl_neumann	164
8.3.2.72	__cyl_neumann_n	164
8.3.2.73	__dawson	166
8.3.2.74	__dawson_cont_frac	166
8.3.2.75	__dawson_series	166
8.3.2.76	__debye_region	166
8.3.2.77	__dilog	167
8.3.2.78	__dirichlet_beta	167
8.3.2.79	__dirichlet_beta	167
8.3.2.80	__dirichlet_eta	168

8.3.2.81	<code>__dirichlet_eta</code>	169
8.3.2.82	<code>__double_factorial</code>	169
8.3.2.83	<code>__ellint_1</code>	169
8.3.2.84	<code>__ellint_2</code>	170
8.3.2.85	<code>__ellint_3</code>	170
8.3.2.86	<code>__ellint_cel</code>	171
8.3.2.87	<code>__ellint_d</code>	171
8.3.2.88	<code>__ellint_el1</code>	171
8.3.2.89	<code>__ellint_el2</code>	171
8.3.2.90	<code>__ellint_el3</code>	171
8.3.2.91	<code>__ellint_rc</code>	171
8.3.2.92	<code>__ellint_rd</code>	172
8.3.2.93	<code>__ellint_rf</code>	172
8.3.2.94	<code>__ellint_rg</code>	173
8.3.2.95	<code>__ellint_rj</code>	173
8.3.2.96	<code>__ellnome</code>	174
8.3.2.97	<code>__ellnome_k</code>	174
8.3.2.98	<code>__ellnome_series</code>	174
8.3.2.99	<code>__expint</code>	174
8.3.2.100	<code>__expint</code>	175
8.3.2.101	<code>__expint_asymp</code>	175
8.3.2.102	<code>__expint_E1</code>	176
8.3.2.103	<code>__expint_E1_asymp</code>	176
8.3.2.104	<code>__expint_E1_series</code>	176
8.3.2.105	<code>__expint_Ei</code>	177
8.3.2.106	<code>__expint_Ei_asymp</code>	177
8.3.2.107	<code>__expint_Ei_series</code>	178
8.3.2.108	<code>__expint_En_cont_frac</code>	178
8.3.2.109	<code>__expint_En_recursion</code>	178
8.3.2.110	<code>__expint_En_series</code>	179
8.3.2.111	<code>__expint_large_n</code>	179
8.3.2.112	<code>__factorial</code>	180
8.3.2.113	<code>__fermi_dirac</code>	180
8.3.2.114	<code>__fisher_f_cdf</code>	180
8.3.2.115	<code>__fisher_f_cdfc</code>	181
8.3.2.116	<code>__fock_airy</code>	181
8.3.2.117	<code>__fpequal</code>	181

8.3.2.118	__fpimag	182
8.3.2.119	__fpimag	182
8.3.2.120	__fpreal	182
8.3.2.121	__fpreal	182
8.3.2.122	__fresnel	182
8.3.2.123	__fresnel_cont_frac	183
8.3.2.124	__fresnel_series	183
8.3.2.125	__gamma	183
8.3.2.126	__gamma_cont_frac	183
8.3.2.127	__gamma_l	183
8.3.2.128	__gamma_p	184
8.3.2.129	__gamma_q	184
8.3.2.130	__gamma_series	184
8.3.2.131	__gamma_temme	184
8.3.2.132	__gamma_u	185
8.3.2.133	__gauss	185
8.3.2.134	__gegenbauer_poly	185
8.3.2.135	__hankel	186
8.3.2.136	__hankel_debye	186
8.3.2.137	__hankel_params	187
8.3.2.138	__hankel_uniform	187
8.3.2.139	__hankel_uniform_olver	187
8.3.2.140	__hankel_uniform_outer	188
8.3.2.141	__hankel_uniform_sum	188
8.3.2.142	__heuman_lambda	188
8.3.2.143	__hurwitz_zeta	189
8.3.2.144	__hurwitz_zeta	189
8.3.2.145	__hurwitz_zeta_euler_maclaurin	189
8.3.2.146	__hydrogen	190
8.3.2.147	__hyperg	190
8.3.2.148	__hyperg_luke	190
8.3.2.149	__hyperg_reflect	190
8.3.2.150	__hyperg_series	191
8.3.2.151	__jacobi_sncndn	191
8.3.2.152	__jacobi_zeta	192
8.3.2.153	__laguerre	192
8.3.2.154	__legendre_q	192

8.3.2.155	<code>__log_bincoef</code>	192
8.3.2.156	<code>__log_double_factorial</code>	193
8.3.2.157	<code>__log_double_factorial</code>	193
8.3.2.158	<code>__log_factorial</code>	193
8.3.2.159	<code>__log_gamma</code>	193
8.3.2.160	<code>__log_gamma_bernoulli</code>	194
8.3.2.161	<code>__log_gamma_lanczos</code>	194
8.3.2.162	<code>__log_gamma_sign</code>	194
8.3.2.163	<code>__log_gamma_spouge</code>	195
8.3.2.164	<code>__log_pochhammer_l</code>	195
8.3.2.165	<code>__log_pochhammer_u</code>	196
8.3.2.166	<code>__logint</code>	196
8.3.2.167	<code>__owens_t</code>	196
8.3.2.168	<code>__pochhammer_l</code>	197
8.3.2.169	<code>__pochhammer_u</code>	197
8.3.2.170	<code>__poly_hermite</code>	197
8.3.2.171	<code>__poly_hermite_asymp</code>	198
8.3.2.172	<code>__poly_hermite_recursion</code>	198
8.3.2.173	<code>__poly_jacobi</code>	199
8.3.2.174	<code>__poly_laguerre</code>	199
8.3.2.175	<code>__poly_laguerre_hypere</code>	199
8.3.2.176	<code>__poly_laguerre_large_n</code>	200
8.3.2.177	<code>__poly_laguerre_recursion</code>	201
8.3.2.178	<code>__poly_legendre_p</code>	201
8.3.2.179	<code>__poly_radial_jacobi</code>	202
8.3.2.180	<code>__polylog</code>	202
8.3.2.181	<code>__polylog</code>	203
8.3.2.182	<code>__polylog_exp</code>	203
8.3.2.183	<code>__polylog_exp_asymp</code>	203
8.3.2.184	<code>__polylog_exp_int_neg</code>	204
8.3.2.185	<code>__polylog_exp_int_neg</code>	204
8.3.2.186	<code>__polylog_exp_int_pos</code>	204
8.3.2.187	<code>__polylog_exp_int_pos</code>	205
8.3.2.188	<code>__polylog_exp_neg</code>	205
8.3.2.189	<code>__polylog_exp_neg</code>	206
8.3.2.190	<code>__polylog_exp_neg_even</code>	207
8.3.2.191	<code>__polylog_exp_neg_odd</code>	207

8.3.2.192	__polylog_exp_negative_real_part	208
8.3.2.193	__polylog_exp_pos	209
8.3.2.194	__polylog_exp_pos	209
8.3.2.195	__polylog_exp_pos	209
8.3.2.196	__polylog_exp_real_neg	210
8.3.2.197	__polylog_exp_real_neg	210
8.3.2.198	__polylog_exp_real_pos	211
8.3.2.199	__polylog_exp_real_pos	211
8.3.2.200	__psi	211
8.3.2.201	__psi	212
8.3.2.202	__psi_asymp	212
8.3.2.203	__psi_series	212
8.3.2.204	__riemann_zeta	213
8.3.2.205	__riemann_zeta_alt	213
8.3.2.206	__riemann_zeta_euler_maclaurin	213
8.3.2.207	__riemann_zeta_glob	213
8.3.2.208	__riemann_zeta_m_1	214
8.3.2.209	__riemann_zeta_m_1_sum	214
8.3.2.210	__riemann_zeta_product	214
8.3.2.211	__riemann_zeta_sum	215
8.3.2.212	__sinc	215
8.3.2.213	__sinc	215
8.3.2.214	__sinc_pi	215
8.3.2.215	__sincosint	216
8.3.2.216	__sincosint_asymp	216
8.3.2.217	__sincosint_cont_frac	216
8.3.2.218	__sincosint_series	216
8.3.2.219	__sinhc	216
8.3.2.220	__sinhc	216
8.3.2.221	__sinhc_pi	217
8.3.2.222	__sinhint	217
8.3.2.223	__sph_bessel	217
8.3.2.224	__sph_bessel	218
8.3.2.225	__sph_bessel_ik	219
8.3.2.226	__sph_bessel_jn	219
8.3.2.227	__sph_hankel	220
8.3.2.228	__sph_hankel_1	221

8.3.2.229	__sph_hankel_1	221
8.3.2.230	__sph_hankel_2	222
8.3.2.231	__sph_hankel_2	222
8.3.2.232	__sph_harmonic	222
8.3.2.233	__sph_legendre	223
8.3.2.234	__sph_neumann	223
8.3.2.235	__sph_neumann	224
8.3.2.236	__student_t_cdf	225
8.3.2.237	__student_t_cdfc	225
8.3.2.238	__theta_1	225
8.3.2.239	__theta_2	226
8.3.2.240	__theta_2_asymp	226
8.3.2.241	__theta_2_sum	226
8.3.2.242	__theta_3	226
8.3.2.243	__theta_3_asymp	227
8.3.2.244	__theta_3_sum	227
8.3.2.245	__theta_4	227
8.3.2.246	__theta_c	227
8.3.2.247	__theta_d	228
8.3.2.248	__theta_n	228
8.3.2.249	__theta_s	228
8.3.2.250	__zernike	228
8.3.2.251	__znorm1	229
8.3.2.252	__znorm2	229
8.3.2.253	evenzeta	229
8.3.3	Variable Documentation	229
8.3.3.1	_Num_Euler_Maclaurin_zeta	229
8.3.3.2	_S_double_factorial_table	229
8.3.3.3	_S_Euler_Maclaurin_zeta	230
8.3.3.4	_S_factorial_table	230
8.3.3.5	_S_neg_double_factorial_table	230
8.3.3.6	_S_num_double_factorials	230
8.3.3.7	_S_num_double_factorials< double >	230
8.3.3.8	_S_num_double_factorials< float >	230
8.3.3.9	_S_num_double_factorials< long double >	230
8.3.3.10	_S_num_factorials	230
8.3.3.11	_S_num_factorials< double >	230

8.3.3.12	_S_num_factorials< float >	230
8.3.3.13	_S_num_factorials< long double >	231
8.3.3.14	_S_num_neg_double_factorials	231
8.3.3.15	_S_num_neg_double_factorials< double >	231
8.3.3.16	_S_num_neg_double_factorials< float >	231
8.3.3.17	_S_num_neg_double_factorials< long double >	231
8.3.3.18	_S_num_zetam1	231
8.3.3.19	_S_zetam1	231
9	Class Documentation	233
9.1	std::__detail::_Factorial_table< _Tp > Struct Template Reference	233
9.1.1	Detailed Description	233
9.1.2	Member Data Documentation	233
9.1.2.1	__factorial	233
9.1.2.2	__log_factorial	233
9.1.2.3	__n	233
10	File Documentation	235
10.1	bits/sf_airy.tcc File Reference	235
10.1.1	Detailed Description	237
10.1.2	Macro Definition Documentation	237
10.1.2.1	_GLIBCXX_BITS_SF_AIRY_TCC	237
10.2	bits/sf_bessel.tcc File Reference	237
10.2.1	Detailed Description	239
10.2.2	Macro Definition Documentation	239
10.2.2.1	_GLIBCXX_BITS_SF_BESSEL_TCC	239
10.3	bits/sf_beta.tcc File Reference	239
10.3.1	Detailed Description	241
10.3.2	Macro Definition Documentation	241
10.3.2.1	_GLIBCXX_BITS_SF_BETA_TCC	241
10.4	bits/sf_cardinal.tcc File Reference	241
10.4.1	Macro Definition Documentation	243
10.4.1.1	_GLIBCXX_BITS_SF_CARDINAL_TCC	243
10.5	bits/sf_chebyshev.tcc File Reference	243
10.5.1	Detailed Description	244
10.5.2	Macro Definition Documentation	244
10.5.2.1	_GLIBCXX_SF_CHEBYSHEV_TCC	244
10.6	bits/sf_dawson.tcc File Reference	245

10.6.1 Detailed Description	246
10.6.2 Macro Definition Documentation	246
10.6.2.1 _GLIBCXX_SF_DAWSON_TCC	246
10.7 bits/sf_ellint.tcc File Reference	246
10.7.1 Detailed Description	248
10.7.2 Macro Definition Documentation	248
10.7.2.1 _GLIBCXX_BITS_SF_ELLINT_TCC	248
10.8 bits/sf_expint.tcc File Reference	249
10.8.1 Detailed Description	251
10.8.2 Macro Definition Documentation	251
10.8.2.1 _GLIBCXX_BITS_SF_EXPINT_TCC	251
10.9 bits/sf_fresnel.tcc File Reference	251
10.9.1 Detailed Description	252
10.9.2 Macro Definition Documentation	252
10.9.2.1 _GLIBCXX_SF_FRESNEL_TCC	252
10.10bits/sf_gamma.tcc File Reference	252
10.10.1 Detailed Description	257
10.10.2 Macro Definition Documentation	258
10.10.2.1 _GLIBCXX_BITS_SF_GAMMA_TCC	258
10.11bits/sf_gegenbauer.tcc File Reference	258
10.11.1 Detailed Description	259
10.11.2 Macro Definition Documentation	259
10.11.2.1 _GLIBCXX_SF_GEGENBAUER_TCC	259
10.12bits/sf_hankel.tcc File Reference	259
10.12.1 Detailed Description	262
10.12.2 Macro Definition Documentation	262
10.12.2.1 _GLIBCXX_BITS_SF_HANKEL_TCC	262
10.13bits/sf_hankel_new.tcc File Reference	262
10.13.1 Macro Definition Documentation	262
10.13.1.1 _GLIBCXX_BITS_SF_HANKEL_NEW_TCC	262
10.14bits/sf_hermite.tcc File Reference	263
10.14.1 Detailed Description	264
10.14.2 Macro Definition Documentation	264
10.14.2.1 _GLIBCXX_BITS_SF_HERMITE_TCC	264
10.15bits/sf_hydrogen.tcc File Reference	264
10.15.1 Detailed Description	265
10.15.2 Macro Definition Documentation	265

10.15.2.1 <code>_GLIBCXX_BITS_SF_HYDROGEN_TCC</code>	265
10.16bits/sf_hyperg.tcc File Reference	265
10.16.1 Detailed Description	267
10.16.2 Macro Definition Documentation	267
10.16.2.1 <code>_GLIBCXX_BITS_SF_HYPERG_TCC</code>	267
10.17bits/sf_hypint.tcc File Reference	267
10.17.1 Detailed Description	268
10.17.2 Macro Definition Documentation	268
10.17.2.1 <code>_GLIBCXX_SF_HYPINT_TCC</code>	268
10.18bits/sf_jacobi.tcc File Reference	269
10.18.1 Detailed Description	270
10.18.2 Macro Definition Documentation	270
10.18.2.1 <code>_GLIBCXX_SF_JACOBI_TCC</code>	270
10.19bits/sf_laguerre.tcc File Reference	270
10.19.1 Detailed Description	272
10.19.2 Macro Definition Documentation	272
10.19.2.1 <code>_GLIBCXX_BITS_SF_LAGUERRE_TCC</code>	272
10.20bits/sf_legendre.tcc File Reference	272
10.20.1 Detailed Description	273
10.20.2 Macro Definition Documentation	273
10.20.2.1 <code>_GLIBCXX_BITS_SF_LEGENDRE_TCC</code>	273
10.21bits/sf_mod_bessel.tcc File Reference	273
10.21.1 Detailed Description	275
10.21.2 Macro Definition Documentation	275
10.21.2.1 <code>_GLIBCXX_BITS_SF_MOD_BESSEL_TCC</code>	275
10.22bits/sf_owens_t.tcc File Reference	276
10.22.1 Detailed Description	276
10.22.2 Macro Definition Documentation	276
10.22.2.1 <code>_GLIBCXX_BITS_SF_OWENS_T_TCC</code>	276
10.23bits/sf_polylog.tcc File Reference	277
10.23.1 Detailed Description	279
10.23.2 Macro Definition Documentation	279
10.23.2.1 <code>_GLIBCXX_BITS_SF_POLYLOG_TCC</code>	279
10.24bits/sf_theta.tcc File Reference	279
10.24.1 Detailed Description	281
10.24.2 Macro Definition Documentation	281
10.24.2.1 <code>_GLIBCXX_SF_THETA_TCC</code>	281

10.25bits/sf_trigint.tcc File Reference	282
10.25.1 Detailed Description	283
10.25.2 Macro Definition Documentation	283
10.25.2.1 _GLIBCXX_SF_TRIGINT_TCC	283
10.26bits/sf_zeta.tcc File Reference	283
10.26.1 Detailed Description	285
10.26.2 Macro Definition Documentation	285
10.26.2.1 _GLIBCXX_BITS_SF_ZETA_TCC	285
10.27bits/specfun.h File Reference	286
10.27.1 Detailed Description	296
10.27.2 Macro Definition Documentation	296
10.27.2.1 __cpp_lib_math_special_functions	296
10.27.2.2 __STDCPP_MATH_SPEC_FUNCS__	297

Index	299
--------------	------------

Chapter 1

Mathematical Special Functions

1.1 Introduction and History

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

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

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

1.2 Contents

The following functions are implemented in namespace `std`:

- [assoc_laguerre](#) - Associated Laguerre functions
- [assoc_legendre](#) - Associated Legendre functions
- [beta](#) - Beta functions
- [comp_ellint_1](#) - Complete elliptic functions of the first kind
- [comp_ellint_2](#) - Complete elliptic functions of the second kind
- [comp_ellint_3](#) - Complete elliptic functions of the third kind
- [cyl_bessel_i](#) - Regular modified cylindrical Bessel functions
- [cyl_bessel_j](#) - Cylindrical Bessel functions of the first kind
- [cyl_bessel_k](#) - Irregular modified cylindrical Bessel functions
- [cyl_neumann](#) - Cylindrical Neumann functions or Cylindrical Bessel functions of the second kind
- [ellint_1](#) - Incomplete elliptic functions of the first kind
- [ellint_2](#) - Incomplete elliptic functions of the second kind
- [ellint_3](#) - Incomplete elliptic functions of the third kind

- [expint](#) - The exponential integral
- [hermite](#) - Hermite polynomials
- [laguerre](#) - Laguerre functions
- [legendre](#) - Legendre polynomials
- [riemann_zeta](#) - The Riemann zeta function
- [sph_bessel](#) - Spherical Bessel functions
- [sph_legendre](#) - Spherical Legendre functions
- [sph_neumann](#) - Spherical Neumann functions

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

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

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

- [airy_ai](#) -
- [airy_bi](#) -
- [bincoef](#) -
- [chebyshev_t](#) -
- [chebyshev_u](#) -
- [chebyshev_v](#) -
- [chebyshev_w](#) -
- [clausen](#) -
- [clausen_c](#) -
- [clausen_s](#) -
- [comp_ellint_d](#) -
- [conf_hyperg_lim](#) -
- [coshint](#) -
- [cosint](#) -
- [cyl_hankel_1](#) -
- [cyl_hankel_2](#) -
- [dawson](#) -
- [dilog](#) -
- [dirichlet_beta](#) -

- [dirichlet_eta](#) -
- [double_factorial](#) -
- [ellint_d](#) -
- [ellint_rc](#) -
- [ellint_rd](#) -
- [ellint_rf](#) -
- [ellint_rg](#) -
- [ellint_rj](#) -

1.3 General Features

1.3.1 Argument Promotion

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

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

1.3.2 NaN Arguments

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

1.4 Implementation

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

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

1.5 Testing

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

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

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

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

1.6 General Bibliography

See also

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

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

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

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

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

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

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

Chapter 2

Todo List

page [Mathematical Special Functions](#)

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

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

this needs some compile-time construction!

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

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

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

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

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

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

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

C++ Mathematical Special Functions	15
C++17/IS29124 Mathematical Special Functions	16
GNU Extended Mathematical Special Functions	38

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

__gnu_cxx	109
std	117
std::__detail	119

Chapter 5

Class Index

5.1 Class List

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

[std::__detail::_Factorial_table<_Tp>](#) 233

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

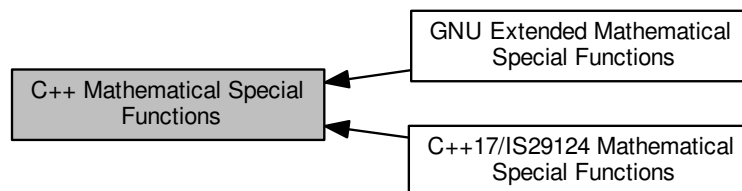
bits/sf_airy.tcc	235
bits/sf_bessel.tcc	237
bits/sf_beta.tcc	239
bits/sf_cardinal.tcc	241
bits/sf_chebyshev.tcc	243
bits/sf_dawson.tcc	245
bits/sf_ellint.tcc	246
bits/sf_expint.tcc	249
bits/sf_fresnel.tcc	251
bits/sf_gamma.tcc	252
bits/sf_gegenbauer.tcc	258
bits/sf_hankel.tcc	259
bits/sf_hankel_new.tcc	262
bits/sf_hermite.tcc	263
bits/sf_hydrogen.tcc	264
bits/sf_hyperg.tcc	265
bits/sf_hypint.tcc	267
bits/sf_jacobi.tcc	269
bits/sf_laguerre.tcc	270
bits/sf_legendre.tcc	272
bits/sf_mod_bessel.tcc	273
bits/sf_owens_t.tcc	276
bits/sf_polylog.tcc	277
bits/sf_theta.tcc	279
bits/sf_trigint.tcc	282
bits/sf_zeta.tcc	283
bits/specfun.h	286

Chapter 7

Module Documentation

7.1 C++ Mathematical Special Functions

Collaboration diagram for C++ Mathematical Special Functions:



Modules

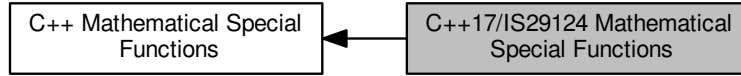
- [C++17/IS29124 Mathematical Special Functions](#)
- [GNU Extended Mathematical Special Functions](#)

7.1.1 Detailed Description

A collection of advanced mathematical special functions.

7.2 C++17/IS29124 Mathematical Special Functions

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



Functions

- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::assoc_laguerre` (unsigned int __n, unsigned int __m, _Tp __x)
- `float std::assoc_laguerref` (unsigned int __n, unsigned int __m, float __x)
- `long double std::assoc_laguerrel` (unsigned int __n, unsigned int __m, long double __x)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::assoc_legendre` (unsigned int __l, unsigned int __m, _Tp __x)
- `float std::assoc_legendref` (unsigned int __l, unsigned int __m, float __x)
- `long double std::assoc_legendrel` (unsigned int __l, unsigned int __m, long double __x)
- `template<typename _Tpa, typename _Tpb >`
`__gnu_cxx::__promote_2< _Tpa, _Tpb >::__type std::beta` (_Tpa __a, _Tpb __b)
- `float std::betaf` (float __a, float __b)
- `long double std::betal` (long double __a, long double __b)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_1` (_Tp __k)
- `float std::comp_ellint_1f` (float __k)
- `long double std::comp_ellint_1l` (long double __k)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_2` (_Tp __k)
- `float std::comp_ellint_2f` (float __k)
- `long double std::comp_ellint_2l` (long double __k)
- `template<typename _Tp, typename _Tpn >`
`__gnu_cxx::__promote_2< _Tp, _Tpn >::__type std::comp_ellint_3` (_Tp __k, _Tpn __nu)
- `float std::comp_ellint_3f` (float __k, float __nu)
- *Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for float modulus k.*
- `long double std::comp_ellint_3l` (long double __k, long double __nu)
- *Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for long double modulus k.*
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_i` (_Tpnu __nu, _Tp __x)
- `float std::cyl_bessel_if` (float __nu, float __x)
- `long double std::cyl_bessel_il` (long double __nu, long double __x)
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type std::cyl_bessel_j` (_Tpnu __nu, _Tp __x)
- `float std::cyl_bessel_jf` (float __nu, float __x)
- `long double std::cyl_bessel_jl` (long double __nu, long double __x)

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

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

7.2.1 Detailed Description

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

7.2.2 Function Documentation

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

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

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

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

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

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

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

where the Laguerre polynomial is defined by:

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

and $x \geq 0$.

See also

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 330 of file `specfun.h`.

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

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

See also

[assoc_laguerre](#) for more details.

Definition at line 282 of file `specfun.h`.

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

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

See also

[assoc_laguerre](#) for more details.

Definition at line 293 of file `specfun.h`.

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

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

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

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

See also

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 378 of file `specfun.h`.

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

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

See also

[assoc_legendre](#) for more details.

Definition at line 345 of file `specfun.h`.

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

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

See also

[assoc_legendre](#) for more details.

Definition at line 356 of file `specfun.h`.

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

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

The beta function is defined by

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

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 423 of file `specfun.h`.

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

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

See also

[beta](#) for more details.

Definition at line 392 of file specfun.h.

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

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

See also

[beta](#) for more details.

Definition at line 402 of file specfun.h.

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

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

The complete elliptic integral of the first kind is defined as

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

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

See also

[ellint_1](#) for details of the incomplete elliptic function of the first kind.

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 471 of file specfun.h.

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

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

See also

[comp_ellint_1](#) for details.

Definition at line 438 of file specfun.h.

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

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

See also

[comp_ellint_1](#) for details.

Definition at line 448 of file specfun.h.

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

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

The complete elliptic integral of the second kind is defined as

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

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

See also

[ellint_2](#) for details of the incomplete elliptic function of the second kind.

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 518 of file specfun.h.

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

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

See also

[comp_ellint_2](#) for details.

Definition at line 486 of file specfun.h.

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

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

See also

[comp_ellint_2](#) for details.

Definition at line 496 of file specfun.h.

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

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

The complete elliptic integral of the third kind is defined as

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

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

See also

[ellint_3](#) for details of the incomplete elliptic function of the third kind.

Template Parameters

<code>_Tp</code>	The floating-point type of the modulus <code>__k</code> .
<code>_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 569 of file `specfun.h`.

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

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

See also

[comp_ellint_3](#) for details.

Definition at line 533 of file `specfun.h`.

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

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

See also

[comp_ellint_3](#) for details.

Definition at line 543 of file `specfun.h`.

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

Return the regular modified Bessel function $I_\nu(x)$ for real order 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 615 of file specfun.h.

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

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

See also

[cyl_bessel_i](#) for details.

Definition at line 584 of file specfun.h.

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

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

See also

[cyl_bessel_i](#) for details.

Definition at line 594 of file specfun.h.

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

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

The cylindrical Bessel function is:

$$J_\nu(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{\nu+2k}}{k! \Gamma(\nu + k + 1)}$$

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 661 of file `specfun.h`.

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

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

See also

[cyl_bessel_j](#) for details.

Definition at line 630 of file `specfun.h`.

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

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

See also

[cyl_bessel_j](#) for details.

Definition at line 640 of file `specfun.h`.

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

Return the irregular modified Bessel function $K_\nu(x)$ of real order 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 713 of file `specfun.h`.

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

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

See also

[cyl_bessel_k](#) for details.

Definition at line 676 of file `specfun.h`.

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

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

See also

[cyl_bessel_k](#) for details.

Definition at line 686 of file `specfun.h`.

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

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

The Neumann function is defined by:

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

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 761 of file `specfun.h`.

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

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

See also

[cyl_neumann](#) for setails.

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

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

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

See also

[cyl_neumann](#) for setails.

Definition at line 738 of file `specfun.h`.

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

Return the incomplete elliptic integral of the first kind $F(k, \phi)$ for `real` modulus k and angle 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 809 of file `specfun.h`.

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

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

See also

[ellint_1](#) for details.

Definition at line 776 of file `specfun.h`.

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

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

See also

[ellint_1](#) for details.

Definition at line 786 of file `specfun.h`.

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

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

The incomplete elliptic integral of the second kind is defined as

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

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

See also

[comp_ellint_2](#).

Template Parameters

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

Parameters

<code>__k</code>	The modulus, <code>abs (__k) <= 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 857 of file `specfun.h`.

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

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

See also

[ellint_2](#) for details.

Definition at line 824 of file `specfun.h`.

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

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

See also

[ellint_2](#) for details.

Definition at line 834 of file `specfun.h`.

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

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

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

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

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

See also

[comp_ellint_3](#).

Template Parameters

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

Parameters

<code>__k</code>	The modulus, <code>abs (__k) <= 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 910 of file `specfun.h`.

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

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

See also

[ellint_3](#) for details.

Definition at line 872 of file `specfun.h`.

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

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

See also

[ellint_3](#) for details.

Definition at line 882 of file `specfun.h`.

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

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

The exponential integral is given by

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

Template Parameters

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

Parameters

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

Definition at line 950 of file `specfun.h`.

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

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

See also

[expint](#) for details.

Definition at line 924 of file `specfun.h`.

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

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

See also

[expint](#) for details.

Definition at line 934 of file `specfun.h`.

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

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

The Hermite polynomial is defined by:

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

The Hermite polynomial obeys a reflection formula:

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

Template Parameters

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

Parameters

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

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

Definition at line 998 of file specfun.h.

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

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

See also

[hermite](#) for details.

Definition at line 965 of file specfun.h.

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

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

See also

[hermite](#) for details.

Definition at line 975 of file specfun.h.

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

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

The Laguerre polynomial is defined by:

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

Template Parameters

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

Parameters

<code>__n</code>	The nonnegative order
<code>__x</code>	The argument <code>__x</code> ≥ 0

Exceptions

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

Definition at line 1042 of file specfun.h.

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

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

See also

[laguerre](#) for more details.

Definition at line 1013 of file specfun.h.

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

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

See also

[laguerre](#) for more details.

Definition at line 1023 of file specfun.h.

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

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

The Legendre function of order l and argument x , $P_l(x)$, is defined by:

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 1087 of file specfun.h.

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

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

See also

[legendre](#) for more details.

Definition at line 1057 of file specfun.h.

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

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

See also

[legendre](#) for more details.

Definition at line 1067 of file specfun.h.

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

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

The Riemann zeta function is defined by:

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

and

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

For $s < 1$ use the reflection formula:

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

Template Parameters

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

Parameters

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

Definition at line 1138 of file specfun.h.

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

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

See also

[riemann_zeta](#) for more details.

Definition at line 1102 of file specfun.h.

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

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

See also

[riemann_zeta](#) for more details.

Definition at line 1112 of file specfun.h.

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

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

The spherical Bessel function is defined by:

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 1182 of file `specfun.h`.

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

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

See also

[sph_bessel](#) for more details.

Definition at line 1153 of file `specfun.h`.

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

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

See also

[sph_bessel](#) for more details.

Definition at line 1163 of file `specfun.h`.

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

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

The spherical Legendre function is defined by

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

Template Parameters

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

Parameters

<code>__l</code>	The order <code>__l >= 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 1229 of file `specfun.h`.

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

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

See also

[sph_legendre](#) for details.

Definition at line 1197 of file `specfun.h`.

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

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

See also

[sph_legendre](#) for details.

Definition at line 1208 of file `specfun.h`.

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

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

The spherical Neumann function is defined by

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

Template Parameters

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

Parameters

<code>__n</code>	The integral order $n \geq 0$
------------------	-------------------------------

<code>__x</code>	The real argument <code>__x >= 0</code>
------------------	--

Exceptions

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

Definition at line 1273 of file `specfun.h`.

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

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

See also

[sph_neumann](#) for details.

Definition at line 1244 of file `specfun.h`.

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

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

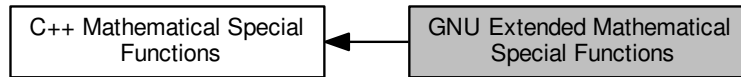
See also

[sph_neumann](#) for details.

Definition at line 1254 of file `specfun.h`.

7.3 GNU Extended Mathematical Special Functions

Collaboration diagram for GNU Extended Mathematical Special Functions:



Enumerations

- enum { [__gnu_cxx::__GLIBCXX_JACOBI_SN](#), [__gnu_cxx::__GLIBCXX_JACOBI_CN](#), [__gnu_cxx::__GLIBCXX_JACOBI_DN](#) }

Functions

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

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

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

- `template<typename _Tp, typename _Tk >`
`__gnu_cxx::__promote_num_t< _Tp, _Tk > __gnu_cxx::ellint_el1 (_Tp __x, _Tk __k_c)`
- `float __gnu_cxx::ellint_el1f (float __x, float __k_c)`
- `long double __gnu_cxx::ellint_el1l (long double __x, long double __k_c)`
- `template<typename _Tp, typename _Tk, typename _Ta, typename _Tb >`
`__gnu_cxx::__promote_num_t< _Tp, _Tk, _Ta, _Tb > __gnu_cxx::ellint_el2 (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)`
- `float __gnu_cxx::ellint_el2f (float __x, float __k_c, float __a, float __b)`
- `long double __gnu_cxx::ellint_el2l (long double __x, long double __k_c, long double __a, long double __b)`
- `template<typename _Tx, typename _Tk, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tx, _Tk, _Tp > __gnu_cxx::ellint_el3 (_Tx __x, _Tk __k_c, _Tp __p)`
- `float __gnu_cxx::ellint_el3f (float __x, float __k_c, float __p)`
- `long double __gnu_cxx::ellint_el3l (long double __x, long double __k_c, long double __p)`
- `template<typename _Tp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Tp, _Up > __gnu_cxx::ellint_rc (_Tp __x, _Up __y)`
- `float __gnu_cxx::ellint_rcf (float __x, float __y)`
- `long double __gnu_cxx::ellint_rcl (long double __x, long double __y)`
- `template<typename _Tp, typename _Up, typename _Vp >`
`__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rd (_Tp __x, _Up __y, _Vp __z)`
- `float __gnu_cxx::ellint_rdf (float __x, float __y, float __z)`
- `long double __gnu_cxx::ellint_rdl (long double __x, long double __y, long double __z)`
- `template<typename _Tp, typename _Up, typename _Vp >`
`__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rf (_Tp __x, _Up __y, _Vp __z)`
- `float __gnu_cxx::ellint_rff (float __x, float __y, float __z)`
- `long double __gnu_cxx::ellint_rfl (long double __x, long double __y, long double __z)`
- `template<typename _Tp, typename _Up, typename _Vp >`
`__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > __gnu_cxx::ellint_rg (_Tp __x, _Up __y, _Vp __z)`
- `float __gnu_cxx::ellint_rgf (float __x, float __y, float __z)`
- `long double __gnu_cxx::ellint_rgl (long double __x, long double __y, long double __z)`
- `template<typename _Tp, typename _Up, typename _Vp, typename _Wp >`
`__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp, _Wp > __gnu_cxx::ellint_rj (_Tp __x, _Up __y, _Vp __z, _Wp __p)`
- `float __gnu_cxx::ellint_rjf (float __x, float __y, float __z, float __p)`
- `long double __gnu_cxx::ellint_rjl (long double __x, long double __y, long double __z, long double __p)`
- `template<typename _Tp >`
`_Tp __gnu_cxx::ellnome (_Tp __k)`
- `float __gnu_cxx::ellnomef (float __k)`
- `long double __gnu_cxx::ellnomel (long double __k)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::expint_e1 (_Tp __x)`
- `float __gnu_cxx::expint_e1f (float __x)`
- `long double __gnu_cxx::expint_e1l (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::expint_en (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::expint_enf (unsigned int __n, float __x)`
- `long double __gnu_cxx::expint_enl (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::factorial (unsigned int __n)`
- `float __gnu_cxx::factorialf (unsigned int __n)`
- `long double __gnu_cxx::factoriall (unsigned int __n)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::fresnel_c (_Tp __x)`

- float [__gnu_cxx::fresnel_cf](#) (float __x)
- long double [__gnu_cxx::fresnel_cl](#) (long double __x)
- template<typename _Tp >
 [__gnu_cxx::promote_num_t](#)< _Tp > [__gnu_cxx::fresnel_s](#) (_Tp __x)
- float [__gnu_cxx::fresnel_sf](#) (float __x)
- long double [__gnu_cxx::fresnel_sl](#) (long double __x)
- template<typename _Tn, typename _Tp >
 [__gnu_cxx::promote_num_t](#)< _Tn, _Tp > [__gnu_cxx::gamma_l](#) (_Tn __n, _Tp __x)
- float [__gnu_cxx::gamma_lf](#) (float __n, float __x)
- long double [__gnu_cxx::gamma_ll](#) (long double __n, long double __x)
- template<typename _Ta, typename _Tp >
 [__gnu_cxx::promote_num_t](#)< _Ta, _Tp > [__gnu_cxx::gamma_p](#) (_Ta __a, _Tp __x)
- 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::promote_num_t](#)< _Ta, _Tp > [__gnu_cxx::gamma_q](#) (_Ta __a, _Tp __x)
- float [__gnu_cxx::gamma_qf](#) (float __a, float __x)
- long double [__gnu_cxx::gamma_ql](#) (long double __a, long double __x)
- template<typename _Tn, typename _Tp >
 [__gnu_cxx::promote_num_t](#)< _Tn, _Tp > [__gnu_cxx::gamma_u](#) (_Tn __n, _Tp __x)
- float [__gnu_cxx::gamma_uf](#) (float __n, float __x)
- long double [__gnu_cxx::gamma_ul](#) (long double __n, long double __x)
- template<typename _Talpha, typename _Tp >
 [__gnu_cxx::promote_num_t](#)< _Talpha, _Tp > [__gnu_cxx::gegenbauer](#) (unsigned int __n, _Talpha __alpha, _Tp __x)
- float [__gnu_cxx::gegenbauerf](#) (unsigned int __n, float __alpha, float __x)
- long double [__gnu_cxx::gegenbauerl](#) (unsigned int __n, long double __alpha, long double __x)
- template<typename _Tk, typename _Tphi >
 [__gnu_cxx::promote_num_t](#)< _Tk, _Tphi > [__gnu_cxx::heuman_lambda](#) (_Tk __k, _Tphi __phi)
- float [__gnu_cxx::heuman_lambdaf](#) (float __k, float __phi)
- long double [__gnu_cxx::heuman_lambdal](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Up >
 [__gnu_cxx::promote_num_t](#)< _Tp, _Up > [__gnu_cxx::hurwitz_zeta](#) (_Tp __s, _Up __a)
- template<typename _Tp, typename _Up >
 std::complex< _Tp > [__gnu_cxx::hurwitz_zeta](#) (_Tp __s, std::complex< _Up > __a)
- float [__gnu_cxx::hurwitz_zetaf](#) (float __s, float __a)
- long double [__gnu_cxx::hurwitz_zetal](#) (long double __s, long double __a)
- template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >
 [__gnu_cxx::promote_4](#)< _Tpa, _Tpb, _Tpc, _Tp >::type [__gnu_cxx::hyperg](#) (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)
- float [__gnu_cxx::hypergf](#) (float __a, float __b, float __c, float __x)
- long double [__gnu_cxx::hypergl](#) (long double __a, long double __b, long double __c, long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
 [__gnu_cxx::promote_num_t](#)< _Ta, _Tb, _Tp > [__gnu_cxx::ibeta](#) (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta, typename _Tb, typename _Tp >
 [__gnu_cxx::promote_num_t](#)< _Ta, _Tb, _Tp > [__gnu_cxx::ibetac](#) (_Ta __a, _Tb __b, _Tp __x)
- float [__gnu_cxx::ibetaf](#) (float __a, float __b, float __x)
- long double [__gnu_cxx::ibetac](#) (long double __a, long double __b, long double __x)
- float [__gnu_cxx::ibetaf](#) (float __a, float __b, float __x)
- long double [__gnu_cxx::ibetal](#) (long double __a, long double __b, long double __x)

- `template<typename _Talpha, typename _Tbeta, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi (unsigned __n, _Talpha __alpha,`
`_Tbeta __beta, _Tp __x)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_cn (_Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_cnf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_cnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_dn (_Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_dnf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_dnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_sn (_Kp __k, _Up __u)`
- `float __gnu_cxx::jacobi_snf (float __k, float __u)`
- `long double __gnu_cxx::jacobi_snl (long double __k, long double __u)`
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::jacobi_zeta (_Tk __k, _Tphi __phi)`
- `float __gnu_cxx::jacobi_zetaf (float __k, float __phi)`
- `long double __gnu_cxx::jacobi_zetal (long double __k, long double __phi)`
- `float __gnu_cxx::jacobif (unsigned __n, float __alpha, float __beta, float __x)`
- `long double __gnu_cxx::jacobil (unsigned __n, long double __alpha, long double __beta, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lbincoef (unsigned int __n, unsigned int __k)`
- `float __gnu_cxx::lbincoeff (unsigned int __n, unsigned int __k)`
- `long double __gnu_cxx::lbincoefl (unsigned int __n, unsigned int __k)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::ldouble_factorial (int __n)`
- `float __gnu_cxx::ldouble_factorialf (int __n)`
- `long double __gnu_cxx::ldouble_factoriall (int __n)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::legendre_q (unsigned int __n, _Tp __x)`
- `float __gnu_cxx::legendre_qf (unsigned int __n, float __x)`
- `long double __gnu_cxx::legendre_ql (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lfactorial (unsigned int __n)`
- `float __gnu_cxx::lfactorialf (unsigned int __n)`
- `long double __gnu_cxx::lfactoriall (unsigned int __n)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::logint (_Tp __x)`
- `float __gnu_cxx::logintf (float __x)`
- `long double __gnu_cxx::logintl (long double __x)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_l (_Tp __a, _Tn __n)`
- `float __gnu_cxx::lpochhammer_lf (float __a, float __n)`
- `long double __gnu_cxx::lpochhammer_ll (long double __a, long double __n)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_u (_Tp __a, _Tn __n)`
- `float __gnu_cxx::lpochhammer_uf (float __a, float __n)`
- `long double __gnu_cxx::lpochhammer_ul (long double __a, long double __n)`
- `template<typename _Tph, typename _Tpa >`
`__gnu_cxx::__promote_num_t< _Tph, _Tpa > __gnu_cxx::owens_t (_Tph __h, _Tpa __a)`
- `float __gnu_cxx::owens_tf (float __h, float __a)`

- long double [__gnu_cxx::owens_tl](#) (long double __h, long double __a)
- template<typename _Tp, typename _Tn >
 [__gnu_cxx::__promote_num_t<_Tp, _Tn > __gnu_cxx::pochhammer_l](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::pochhammer_lf](#) (float __a, float __n)
- long double [__gnu_cxx::pochhammer_ll](#) (long double __a, long double __n)
- template<typename _Tp, typename _Tn >
 [__gnu_cxx::__promote_num_t<_Tp, _Tn > __gnu_cxx::pochhammer_u](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::pochhammer_uf](#) (float __a, float __n)
- long double [__gnu_cxx::pochhammer_ul](#) (long double __a, long double __n)
- template<typename _Tp, typename _Wp >
 [__gnu_cxx::__promote_num_t<_Tp, _Wp > __gnu_cxx::polylog](#) (_Tp __s, _Wp __w)
- template<typename _Tp, typename _Wp >
 std::complex< [__gnu_cxx::__promote_num_t<_Tp, _Wp > __gnu_cxx::polylog](#) (_Tp __s, std::complex< _Tp > __w)
- float [__gnu_cxx::polylogf](#) (float __s, float __w)
- std::complex< float > [__gnu_cxx::polylogf](#) (float __s, std::complex< float > __w)
- long double [__gnu_cxx::polylogl](#) (long double __s, long double __w)
- std::complex< long double > [__gnu_cxx::polylogl](#) (long double __s, std::complex< long double > __w)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::psi](#) (_Tp __x)
- float [__gnu_cxx::psif](#) (float __x)
- long double [__gnu_cxx::psil](#) (long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::radpoly](#) (unsigned int __n, unsigned int __m, _Tp __rho)
- float [__gnu_cxx::radpolyf](#) (unsigned int __n, unsigned int __m, float __rho)
- long double [__gnu_cxx::radpolyl](#) (unsigned int __n, unsigned int __m, long double __rho)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sinc](#) (_Tp __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sinc_pi](#) (_Tp __x)
- float [__gnu_cxx::sinc_pif](#) (float __x)
- long double [__gnu_cxx::sinc_pil](#) (long double __x)
- float [__gnu_cxx::sincf](#) (float __x)
- long double [__gnu_cxx::sincl](#) (long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sinhc](#) (_Tp __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sinhc_pi](#) (_Tp __x)
- float [__gnu_cxx::sinhc_pif](#) (float __x)
- long double [__gnu_cxx::sinhc_pil](#) (long double __x)
- float [__gnu_cxx::sinhcf](#) (float __x)
- long double [__gnu_cxx::sinhcl](#) (long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sinhint](#) (_Tp __x)
- float [__gnu_cxx::sinhintf](#) (float __x)
- long double [__gnu_cxx::sinhintl](#) (long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sinint](#) (_Tp __x)
- float [__gnu_cxx::sinintf](#) (float __x)
- long double [__gnu_cxx::sinintl](#) (long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sph_bessel_i](#) (unsigned int __n, _Tp __x)

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

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

7.3.1 Detailed Description

An extended collection of advanced mathematical special functions for GNU.

7.3.2 Enumeration Type Documentation

7.3.2.1 anonymous enum

Enumerator

```
__GLIBCXX_JACOBI_SN
__GLIBCXX_JACOBI_CN
__GLIBCXX_JACOBI_DN
```

Definition at line 1693 of file specfun.h.

7.3.3 Function Documentation

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

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

The Airy function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 2632 of file specfun.h.

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

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

See also

[airy_ai](#) for details.

Definition at line 2597 of file specfun.h.

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

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

See also

[airy_ai](#) for details.

Definition at line 2611 of file specfun.h.

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

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

The Airy function is defined by:

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

Template Parameters

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

Parameters

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

Definition at line 2684 of file specfun.h.

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

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

See also

[airy_bi](#) for details.

Definition at line 2648 of file specfun.h.

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

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

See also

[airy_bi](#) for details.

Definition at line 2662 of file `specfun.h`.

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

Return the Bernoulli number of integer order n .

The Bernoulli numbers are defined by

Parameters

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

Definition at line 3660 of file `specfun.h`.

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

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

See also

[bernoulli](#) for details.

Definition at line 3635 of file `specfun.h`.

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

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

See also

[bernoulli](#) for details.

Definition at line 3645 of file `specfun.h`.

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

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

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

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

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

Definition at line 3592 of file specfun.h.

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

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

The Chebyshev polynomial of the first kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 1891 of file specfun.h.

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

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

See also

[chebyshev_t](#) for details.

Definition at line 1862 of file specfun.h.

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

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

See also

[chebyshev_t](#) for details.

Definition at line 1872 of file specfun.h.

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

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

The Chebyshev polynomial of the second kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 1935 of file specfun.h.

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

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

See also

[chebyshev_u](#) for details.

Definition at line 1906 of file specfun.h.

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

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

See also

[chebyshev_u](#) for details.

Definition at line 1916 of file specfun.h.

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

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

The Chebyshev polynomial of the third kind is defined by:

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

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

Template Parameters

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

Parameters

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

Definition at line 1980 of file specfun.h.

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

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

See also

[chebyshev_v](#) for details.

Definition at line 1950 of file `specfun.h`.

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

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

See also

[chebyshev_v](#) for details.

Definition at line 1960 of file `specfun.h`.

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

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

The Chebyshev polynomial of the fourth kind is defined by:

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

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

Template Parameters

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

Parameters

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

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

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

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

See also

[chebyshev_w](#) for details.

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

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

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

See also

[chebyshev_w](#) for details.

Definition at line 2005 of file specfun.h.

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

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

The Clausen function is defined by

Parameters

<code>__m</code>	
<code>__w</code>	The complex argument

Definition at line 4603 of file specfun.h.

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

Definition at line 4624 of file specfun.h.

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

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

The Clausen cosine function is defined by

Parameters

<code>__m</code>	
<code>__w</code>	

Definition at line 4564 of file specfun.h.

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

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

See also

[clausen_c](#) for details.

Definition at line 4539 of file specfun.h.

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

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

See also

[clausen_c](#) for details.

Definition at line 4548 of file `specfun.h`.

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

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

The Clausen sine function is defined by

Parameters

<code>__m</code>	
<code>__w</code>	

Definition at line 4525 of file `specfun.h`.

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

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

See also

[clausen_s](#) for details.

Definition at line 4500 of file `specfun.h`.

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

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

See also

[clausen_s](#) for details.

Definition at line 4509 of file `specfun.h`.

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

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

See also

[clausen](#) for details.

Definition at line 4578 of file `specfun.h`.

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

Definition at line 4612 of file specfun.h.

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

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

See also

[clausen](#) for details.

Definition at line 4587 of file specfun.h.

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

Definition at line 4616 of file specfun.h.

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

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

The complete Legendre elliptic integral D is defined by

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

Template Parameters

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

Parameters

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

Definition at line 3846 of file specfun.h.

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

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

See also

[comp_ellint_d](#) for details.

Definition at line 3819 of file specfun.h.

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

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

See also

[comp_ellint_d](#) for details.

Definition at line 3829 of file specfun.h.

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

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

See also

[comp_ellint_rf](#) for details.

Definition at line 2805 of file specfun.h.

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

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

See also

[comp_ellint_rf](#) for details.

Definition at line 2815 of file specfun.h.

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

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

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

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

Parameters

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

Definition at line 2833 of file specfun.h.

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

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

See also

[comp_ellint_rg](#) for details.

Definition at line 3038 of file specfun.h.

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

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

See also

[comp_ellint_rg](#) for details.

Definition at line 3047 of file specfun.h.

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

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

The complete Carlson elliptic function is defined by:

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

Parameters

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

Definition at line 3066 of file specfun.h.

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

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

The confluent hypergeometric function is defined by

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

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

Parameters

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

Definition at line 1336 of file specfun.h.

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

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

The confluent hypergeometric limit function is defined by

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

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

Parameters

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

Definition at line 1432 of file specfun.h.

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

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

See also

[conf_hyperg_lim](#) for details.

Definition at line 1403 of file specfun.h.

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

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

See also

[conf_hyperg_lim](#) for details.

Definition at line 1413 of file specfun.h.

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

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

See also

[conf_hyperg](#) for details.

Definition at line 1304 of file specfun.h.

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

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

See also

[conf_hyperg](#) for details.

Definition at line 1315 of file specfun.h.

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

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

The hyperbolic cosine integral is defined by

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

Template Parameters

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

Parameters

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

Definition at line 1686 of file `specfun.h`.

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

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

See also

[coshint](#) for details.

Definition at line 1658 of file `specfun.h`.

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

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

See also

[coshint](#) for details.

Definition at line 1668 of file `specfun.h`.

7.3.3.55 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::cosint (_Tp __x) [inline]`

Return the cosine integral $Ci(x)$ 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 1603 of file specfun.h.

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

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

See also

[cosint](#) for details.

Definition at line 1577 of file specfun.h.

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

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

See also

[cosint](#) for details.

Definition at line 1587 of file specfun.h.

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

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

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

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

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

See also

`cyl_bessel` and `cyl_neumann`).

Template Parameters

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

Parameters

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

Definition at line 2334 of file specfun.h.

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

Return the complex cylindrical Hankel function of the first kind $H_\nu^{(1)}(x)$ of complex order 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 4123 of file `specfun.h`.

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

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

See also

[cyl_hankel_1](#) for details.

Definition at line 2301 of file `specfun.h`.

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

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

See also

[cyl_hankel_1](#) for more details.

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

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

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

See also

[cyl_hankel_1](#) for details.

Definition at line 2312 of file `specfun.h`.

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

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

See also

[cyl_hankel_1](#) for more details.

Definition at line 4103 of file `specfun.h`.

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

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

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

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

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

See also

`cyl_bessel` and `cyl_neumann`).

Template Parameters

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

Parameters

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

Definition at line 2383 of file `specfun.h`.

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

Return the complex cylindrical Hankel function of the second kind $H_\nu^{(2)}(x)$ of complex order 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 4170 of file `specfun.h`.

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

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

See also

[cyl_hankel_2](#) for details.

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

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

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

See also

[cyl_hankel_2](#) for more details.

Definition at line 4139 of file `specfun.h`.

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

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

See also

[cyl_hankel_2](#) for details.

Definition at line 2361 of file `specfun.h`.

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

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

See also

[cyl_hankel_2](#) for more details.

Definition at line 4150 of file `specfun.h`.

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

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

The Dawson integral is defined by:

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

and it's derivative is:

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

Parameters

<code>__x</code>	The argument $-\text{inf} < x < \text{inf}$.
------------------	---

Definition at line 3376 of file `specfun.h`.

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

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

See also

[dawson](#) for details.

Definition at line 3348 of file `specfun.h`.

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

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

See also

[dawson](#) for details.

Definition at line 3357 of file `specfun.h`.

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

Definition at line 2749 of file `specfun.h`.

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

Definition at line 2737 of file `specfun.h`.

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

Definition at line 2741 of file `specfun.h`.

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

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

The dilogarithm is defined by:

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

Parameters

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

Definition at line 2790 of file specfun.h.

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

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

See also

[dilog](#) for details.

Definition at line 2764 of file specfun.h.

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

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

See also

[dilog](#) for details.

Definition at line 2774 of file specfun.h.

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

Return the Dirichlet beta function of real argument s .

The Dirichlet beta function is defined by:

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

An important reflection formula is:

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

Parameters

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

Definition at line 4486 of file specfun.h.

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

Return the Dirichlet beta function of real argument s .

See also

[dirichlet_beta](#) for details.

Definition at line 4457 of file specfun.h.

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

Return the Dirichlet beta function of real argument s .

See also

[dirichlet_beta](#) for details.

Definition at line 4466 of file specfun.h.

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

Return the Dirichlet eta function of real argument s .

The Dirichlet eta function is defined by

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

An important reflection formula is:

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

Parameters

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

Definition at line 4443 of file specfun.h.

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

Return the Dirichlet eta function of real argument s .

See also

[dirichlet_eta](#) for details.

Definition at line 4413 of file specfun.h.

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

Return the Dirichlet eta function of real argument s .

See also

[dirichlet_eta](#) for details.

Definition at line 4422 of file specfun.h.

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

Definition at line 3537 of file specfun.h.

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

Definition at line 3525 of file specfun.h.

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

Definition at line 3529 of file specfun.h.

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

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

The Bulirsch complete elliptic integral is defined by

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

Parameters

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

Definition at line 4076 of file specfun.h.

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

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

See also

[ellint_cel](#) for details.

Definition at line 4044 of file specfun.h.

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

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

See also

[ellint_cel](#) for details.

Definition at line 4053 of file specfun.h.

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

Return the incomplete Legendre elliptic integral $D(k, \phi)$ of real modulus k and angular limit 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 3889 of file specfun.h.

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

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

See also

[ellint_d](#) for details.

Definition at line 3861 of file specfun.h.

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

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

See also

[ellint_d](#) for details.

Definition at line 3871 of file specfun.h.

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

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

The Bulirsch elliptic integral of the first kind is defined by

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

Parameters

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

Definition at line 3935 of file `specfun.h`.

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

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

See also

[ellint_el1](#) for details.

Definition at line 3905 of file `specfun.h`.

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

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

See also

[ellint_el1](#) for details.

Definition at line 3916 of file `specfun.h`.

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

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

The Bulirsch elliptic integral of the second kind is defined by

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

Parameters

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

Definition at line 3981 of file `specfun.h`.

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

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

See also

[ellint_el2](#) for details.

Definition at line 3950 of file specfun.h.

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

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

See also

[ellint_el2](#) for details.

Definition at line 3960 of file specfun.h.

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

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

The Bulirsch elliptic integral of the third kind is defined by

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

Parameters

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

Definition at line 4028 of file specfun.h.

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

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

See also

[ellint_el3](#) for details.

Definition at line 3997 of file specfun.h.

7.3.3.102 `long double __gnu_cxx::ellint_el3(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 4008 of file specfun.h.

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

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

The Carlson elliptic function is defined by:

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 2925 of file specfun.h.

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

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

See also

[ellint_rc](#) for details.

Definition at line 2891 of file specfun.h.

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

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

See also

[ellint_rc](#) for details.

Definition at line 2900 of file specfun.h.

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

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

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

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

Based on Carlson's algorithms:

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

Parameters

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

Definition at line 3024 of file specfun.h.

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

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

See also

[ellint_rd](#) for details.

Definition at line 2988 of file specfun.h.

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

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

See also

[ellint_rd](#) for details.

Definition at line 2997 of file specfun.h.

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

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

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

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

Parameters

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

Definition at line 2877 of file `specfun.h`.

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

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

See also

[ellint_rf](#) for details.

Definition at line 2848 of file `specfun.h`.

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

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

See also

[ellint_rf](#) for details.

Definition at line 2858 of file `specfun.h`.

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

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

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

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

Based on Carlson's algorithms:

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

Parameters

<code>__x</code>	The first of three symmetric arguments.
------------------	---

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

Definition at line 3115 of file specfun.h.

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

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

See also

[ellint_rg](#) for details.

Definition at line 3080 of file specfun.h.

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

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

See also

[ellint_rg](#) for details.

Definition at line 3089 of file specfun.h.

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

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

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

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

Based on Carlson's algorithms:

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

Parameters

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

<code>__p</code>	The fourth argument.
------------------	----------------------

Definition at line 2974 of file specfun.h.

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

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

See also

[ellint_rj](#) for details.

Definition at line 2939 of file specfun.h.

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

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

See also

[ellint_rj](#) for details.

Definition at line 2948 of file specfun.h.

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

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

The elliptic nome function is defined by

$$q(k) =$$

Parameters

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

Definition at line 4836 of file specfun.h.

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

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

See also

[ellnome](#) for details.

Definition at line 4811 of file specfun.h.

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

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

See also

[ellnome](#) for details.

Definition at line 4821 of file specfun.h.

7.3.3.121 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::expint_e1 (_Tp __x) [inline]`

Definition at line 3392 of file specfun.h.

7.3.3.122 `float __gnu_cxx::expint_e1f (float __x) [inline]`

Definition at line 3383 of file specfun.h.

7.3.3.123 `long double __gnu_cxx::expint_e1l (long double __x) [inline]`

Definition at line 3387 of file specfun.h.

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

Definition at line 3411 of file specfun.h.

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

Definition at line 3399 of file specfun.h.

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

Definition at line 3403 of file specfun.h.

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

Definition at line 3516 of file specfun.h.

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

Definition at line 3504 of file specfun.h.

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

Definition at line 3508 of file specfun.h.

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

Return the Fresnel cosine integral of argument x .

The Fresnel cosine integral is defined by

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

Parameters

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

Definition at line 3334 of file `specfun.h`.

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

Definition at line 3315 of file `specfun.h`.

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

Definition at line 3319 of file `specfun.h`.

7.3.3.133 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::fresnel_s(_Tp __x) [inline]`

Return the Fresnel sine integral of argument x .

The Fresnel sine integral is defined by

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

Parameters

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

Definition at line 3306 of file `specfun.h`.

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

Definition at line 3287 of file `specfun.h`.

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

Definition at line 3291 of file `specfun.h`.

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

Definition at line 2728 of file `specfun.h`.

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

Definition at line 2716 of file `specfun.h`.

7.3.3.138 `long double __gnu_cxx::gamma_l(long double __n, long double __x)` `[inline]`

Definition at line 2720 of file specfun.h.

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

Definition at line 3714 of file specfun.h.

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

Definition at line 3702 of file specfun.h.

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

Definition at line 3706 of file specfun.h.

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

Definition at line 3735 of file specfun.h.

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

Definition at line 3723 of file specfun.h.

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

Definition at line 3727 of file specfun.h.

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

Definition at line 2707 of file specfun.h.

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

Definition at line 2695 of file specfun.h.

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

Definition at line 2699 of file specfun.h.

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

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

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

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

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

Template Parameters

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

Parameters

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

Definition at line 2133 of file `specfun.h`.

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

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

See also

[gegenbauer](#) for details.

Definition at line 2100 of file `specfun.h`.

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

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

See also

[gegenbauer](#) for details.

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

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

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

The complete Heuman lambda function is defined by

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

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

Template Parameters

<code>__Tk</code>	the floating-point type of the modulus
<code>__Tphi</code>	the floating-point type of the angular limit argument

Parameters

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

Definition at line 3804 of file `specfun.h`.

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

Definition at line 3778 of file `specfun.h`.

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

Definition at line 3782 of file `specfun.h`.

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

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

The the Hurwitz zeta function is defined by

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

Parameters

<code>__s</code>	The argument
<code>__a</code>	The parameter

Definition at line 3156 of file `specfun.h`.

7.3.3.155 `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 3170 of file `specfun.h`.

7.3.3.156 `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 3130 of file specfun.h.

7.3.3.157 `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 3140 of file specfun.h.

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

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

The hypergeometric function is defined by

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

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

Parameters

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

Definition at line 1385 of file specfun.h.

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

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

See also

[hyperg](#) for details.

Definition at line 1352 of file specfun.h.

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

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

See also

[hyperg](#) for details.

Definition at line 1363 of file specfun.h.

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

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

The regularized incomplete beta function is defined by

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

where

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

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

Parameters

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

Definition at line 3247 of file specfun.h.

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

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

The regularized complementary incomplete beta function is defined by

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

Parameters

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

Definition at line 3278 of file specfun.h.

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

Definition at line 3256 of file specfun.h.

References `__gnu_cxx::ibetaf()`.

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

Definition at line 3260 of file specfun.h.

References `__gnu_cxx::ibetal()`.

7.3.3.165 `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 3213 of file `specfun.h`.

Referenced by `__gnu_cxx::ibetaf()`.

7.3.3.166 `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 3223 of file `specfun.h`.

Referenced by `__gnu_cxx::ibetac()`.

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

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

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

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

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

Template Parameters

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

Parameters

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

Definition at line 2085 of file `specfun.h`.

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

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

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

The Jacobi elliptic `cn` integral is defined by

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 1795 of file `specfun.h`.

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

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

See also

[jacobi_cn](#) for details.

Definition at line 1760 of file `specfun.h`.

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

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

See also

[jacobi_cn](#) for details.

Definition at line 1773 of file `specfun.h`.

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

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

The Jacobi elliptic dn integral is defined by

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 1846 of file `specfun.h`.

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

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

See also

[jacobi_dn](#) for details.

Definition at line 1811 of file specfun.h.

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

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

See also

[jacobi_dn](#) for details.

Definition at line 1824 of file specfun.h.

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

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

The Jacobi elliptic `sn` integral is defined by

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 1744 of file specfun.h.

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

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

See also

[jacobi_sn](#) for details.

Definition at line 1709 of file specfun.h.

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

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

See also

[jacobi_sn](#) for details.

Definition at line 1722 of file specfun.h.

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

Return the Jacobi zeta function of k and 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 3769 of file `specfun.h`.

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

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

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

Definition at line 3748 of file `specfun.h`.

7.3.3.180 `float __gnu_cxx::jacobiP (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 2041 of file `specfun.h`.

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

7.3.3.181 `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 2052 of file `specfun.h`.

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

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

Definition at line 3621 of file specfun.h.

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

Definition at line 3609 of file specfun.h.

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

Definition at line 3613 of file specfun.h.

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

Definition at line 3579 of file specfun.h.

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

Definition at line 3567 of file specfun.h.

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

Definition at line 3571 of file specfun.h.

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

Definition at line 3693 of file specfun.h.

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

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

See also

[legendre_q](#) for details.

Definition at line 3675 of file specfun.h.

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

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

See also

[legendre_q](#) for details.

Definition at line 3685 of file specfun.h.

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

Definition at line 3558 of file specfun.h.

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

Definition at line 3546 of file specfun.h.

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

Definition at line 3550 of file specfun.h.

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

Return the logarithmic integral of argument x .

The logarithmic integral is defined by

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

Parameters

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

Definition at line 1524 of file specfun.h.

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

Return the logarithmic integral of argument x .

See also

[logint](#) for details.

Definition at line 1500 of file specfun.h.

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

Return the logarithmic integral of argument x .

See also

[logint](#) for details.

Definition at line 1509 of file specfun.h.

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

Definition at line 3453 of file specfun.h.

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

Definition at line 3441 of file specfun.h.

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

Definition at line 3445 of file specfun.h.

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

Definition at line 3432 of file specfun.h.

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

Definition at line 3420 of file specfun.h.

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

Definition at line 3424 of file specfun.h.

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

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

The Owens T function is defined by

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

Parameters

<code>__h</code>	The shape factor
<code>__a</code>	The integration limit

Definition at line 5047 of file specfun.h.

7.3.3.204 `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 5019 of file specfun.h.

7.3.3.205 `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 5029 of file `specfun.h`.

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

Definition at line 3495 of file `specfun.h`.

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

Definition at line 3483 of file `specfun.h`.

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

Definition at line 3487 of file `specfun.h`.

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

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

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

Definition at line 3462 of file `specfun.h`.

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

Definition at line 3466 of file `specfun.h`.

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

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

The polylogarithm function is defined by

Parameters

<code>__s</code>	
<code>__w</code>	

Definition at line 4359 of file specfun.h.

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

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

The polylogarithm function is defined by

Parameters

<code>__s</code>	
<code>__w</code>	

Definition at line 4399 of file specfun.h.

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

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

See also

[polylog](#) for details.

Definition at line 4332 of file specfun.h.

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

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

See also

[polylog](#) for details.

Definition at line 4372 of file specfun.h.

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

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

See also

[polylog](#) for details.

Definition at line 4342 of file specfun.h.

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

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

See also

[polylog](#) for details.

Definition at line 4382 of file `specfun.h`.

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

Return the psi or digamma function of argument x .

The the psi or digamma function is defined by

$$\psi(x) =$$

Parameters

<code>__x</code>	The parameter
------------------	---------------

Definition at line 3198 of file `specfun.h`.

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

Definition at line 3179 of file `specfun.h`.

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

Definition at line 3183 of file `specfun.h`.

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

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

The radial polynomials are defined by

$$R_n^m(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k! (\frac{n+m}{2} - k)! (\frac{n-m}{2} - k)!} \rho^{n-2k}$$

for $n - m$ even and identically 0 for $n - m$ odd. The radial polynomials can be related to the Jacobi polynomials:

$$R_n^m(\rho) =$$

See also

[jacobi](#) for details on the Jacobi polynomials.

Template Parameters

<code>__Tp</code>	The real type of the radial coordinate
-------------------	--

Parameters

<code>__n</code>	The non-negative degree.
<code>__m</code>	The non-negative azimuthal order
<code>__rho</code>	The radial argument

Definition at line 2243 of file `specfun.h`.

7.3.3.222 `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 2204 of file `specfun.h`.

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

7.3.3.223 `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 2215 of file `specfun.h`.

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

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

Definition at line 1486 of file `specfun.h`.

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

Definition at line 1459 of file `specfun.h`.

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

Definition at line 1444 of file `specfun.h`.

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

Definition at line 1451 of file `specfun.h`.

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

Definition at line 1471 of file `specfun.h`.

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

Definition at line 1478 of file `specfun.h`.

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

Definition at line 2285 of file `specfun.h`.

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

Definition at line 2264 of file `specfun.h`.

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

Definition at line 2252 of file `specfun.h`.

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

Definition at line 2256 of file `specfun.h`.

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

Definition at line 2273 of file `specfun.h`.

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

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

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

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

The hyperbolic sine integral is defined by

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

Template Parameters

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

Parameters

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

Definition at line 1644 of file specfun.h.

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

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

See also

[sinhint](#) for details.

Definition at line 1617 of file specfun.h.

7.3.3.238 `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 1627 of file specfun.h.

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

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

The sine integral is defined by

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

Parameters

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

Definition at line 1563 of file specfun.h.

7.3.3.240 `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 1538 of file specfun.h.

7.3.3.241 `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 1548 of file `specfun.h`.

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

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

The spherical Bessel function is defined by:

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 2523 of file `specfun.h`.

7.3.3.243 `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 2484 of file `specfun.h`.

7.3.3.244 `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 2499 of file `specfun.h`.


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

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

The spherical Bessel function is defined by:

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

Template Parameters

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

Parameters

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

Exceptions

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

Definition at line 2580 of file `specfun.h`.

```
7.3.3.246  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 2541 of file `specfun.h`.

```
7.3.3.247  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 2556 of file `specfun.h`.

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

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

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

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

Template Parameters

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

Parameters

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

Definition at line 2426 of file `specfun.h`.

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

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

The spherical Hankel function of the first kind is defined by

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

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

Parameters

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

Definition at line 4218 of file `specfun.h`.

```
7.3.3.250 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 2398 of file `specfun.h`.

```
7.3.3.251 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 4186 of file `specfun.h`.

```
7.3.3.252 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 2408 of file specfun.h.

```
7.3.3.253 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 4197 of file specfun.h.

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

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

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

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

Template Parameters

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

Parameters

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

Definition at line 2469 of file specfun.h.

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

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

The spherical Hankel function of the second kind is defined by

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

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

Parameters

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

Definition at line 4266 of file `specfun.h`.

7.3.3.256 `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 2441 of file `specfun.h`.

7.3.3.257 `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 4234 of file `specfun.h`.

7.3.3.258 `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 2451 of file `specfun.h`.

7.3.3.259 `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 4245 of file `specfun.h`.

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

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

The spherical harmonic function is defined by:

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

Parameters

<code>__l</code>	The order
<code>__m</code>	The degree
<code>__theta</code>	The zenith angle in radians
<code>__phi</code>	The azimuth angle in radians

Definition at line 4317 of file specfun.h.

```
7.3.3.261  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 4281 of file specfun.h.

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

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

See also

[sph_harmonic](#) for details.

Definition at line 4293 of file specfun.h.

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

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

The Neville theta-1 function is defined by

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

Parameters

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

Definition at line 4667 of file specfun.h.

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

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

See also

[theta_1](#) for details.

Definition at line 4639 of file specfun.h.

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

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

See also

[theta_1](#) for details.

Definition at line 4649 of file specfun.h.

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

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

The exponential theta-2 function is defined by

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

Parameters

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

Definition at line 4710 of file specfun.h.

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

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

See also

[theta_2](#) for details.

Definition at line 4682 of file specfun.h.

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

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

See also

[theta_2](#) for details.

Definition at line 4692 of file specfun.h.

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

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

The exponential theta-3 function is defined by

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

Parameters

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

Definition at line 4753 of file specfun.h.

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

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

See also

[theta_3](#) for details.

Definition at line 4725 of file specfun.h.

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

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

See also

[theta_3](#) for details.

Definition at line 4735 of file specfun.h.

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

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period nu and argument x .

The exponential theta-4 function is defined by

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

Parameters

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

Definition at line 4796 of file specfun.h.

7.3.3.273 `float __gnu_cxx::theta_4f(float __nu, float __x) [inline]`

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period nu and argument x .

See also

[theta_4](#) for details.

Definition at line 4768 of file specfun.h.

7.3.3.274 `long double __gnu_cxx::theta_4l(long double __nu, long double __x) [inline]`

Return the exponential theta-4 function $\theta_4(\nu, x)$ of period nu and argument x .

See also

[theta_4](#) for details.

Definition at line 4778 of file specfun.h.

7.3.3.275 `template<typename _Tpk, typename _Tp> __gnu_cxx::__promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_c(_Tpk __k, _Tp __x) [inline]`

Return the Neville theta-c function $\theta_c(k, x)$ of modulus k and argument x .

The Neville theta-c function is defined by

Parameters

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

Definition at line 4920 of file specfun.h.

7.3.3.276 `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 4893 of file specfun.h.

7.3.3.277 `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 4903 of file `specfun.h`.

7.3.3.278 `template<typename _Tpk, typename _Tp> __gnu_cxx::__promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_d (_Tpk __k, _Tp __x) [inline]`

Return the Neville theta-d function $\theta_d(k, x)$ of modulus k and argument x .

The Neville theta-d function is defined by

$$\theta_d(k, x) =$$

Parameters

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

Definition at line 4962 of file `specfun.h`.

7.3.3.279 `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 4935 of file `specfun.h`.

7.3.3.280 `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 4945 of file `specfun.h`.

7.3.3.281 `template<typename _Tpk, typename _Tp> __gnu_cxx::__promote_num_t<_Tpk, _Tp> __gnu_cxx::theta_n (_Tpk __k, _Tp __x) [inline]`

Return the Neville theta-n function $\theta_n(k, x)$ of modulus k and argument x .

The Neville theta-n function is defined by

$$\theta_n(k, x) =$$

Parameters

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

Definition at line 5004 of file specfun.h.

7.3.3.282 `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 4977 of file specfun.h.

7.3.3.283 `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 4987 of file specfun.h.

7.3.3.284 `template<typename _Tp, typename _Tp> __gnu_cxx::promote_num_t<_Tp, _Tp> __gnu_cxx::theta_s (_Tp __k, _Tp __x) [inline]`

Return the Neville theta-s function $\theta_s(k, x)$ of modulus k and argument x .

The Neville theta-s function is defined by

Parameters

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

Definition at line 4878 of file specfun.h.

7.3.3.285 `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 4851 of file specfun.h.

7.3.3.286 `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 4861 of file `specfun.h`.

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

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n , signed order m , and real radial argument ρ and azimuthal angle ϕ .

The even Zernicke polynomials are defined by:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

and the odd Zernicke polynomials are defined by:

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative degree m and $m \leq n$ and where $R_n^m(\rho)$ is the radial polynomial (

See also

[radpoly](#)).

Template Parameters

<code>_Trho</code>	The real type of the radial coordinate
<code>_Tphi</code>	The real type of the azimuthal angle

Parameters

<code>__n</code>	The non-negative degree.
<code>__m</code>	The (signed) azimuthal order
<code>__rho</code>	The radial coordinate
<code>__phi</code>	The azimuthal angle

Definition at line 2188 of file `specfun.h`.

7.3.3.288 `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 2149 of file `specfun.h`.

7.3.3.289 `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 2160 of file specfun.h.

Chapter 8

Namespace Documentation

8.1 `__gnu_cxx` Namespace Reference

Enumerations

- enum { [_GLIBCXX_JACOBI_SN](#), [_GLIBCXX_JACOBI_CN](#), [_GLIBCXX_JACOBI_DN](#) }

Functions

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

- long double [chebyshev_vl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [chebyshev_w](#) (unsigned int __n, _Tp __x)
- float [chebyshev_wf](#) (unsigned int __n, float __x)
- long double [chebyshev_wl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [clausen](#) (unsigned int __m, _Tp __w)
- template<typename _Tp >
std::complex< __gnu_cxx::__promote_num_t<_Tp > > [clausen](#) (unsigned int __m, std::complex<_Tp > __w)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [clausen_c](#) (unsigned int __m, _Tp __w)
- float [clausen_cf](#) (unsigned int __m, float __w)
- long double [clausen_cl](#) (unsigned int __m, long double __w)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [clausen_s](#) (unsigned int __m, _Tp __w)
- float [clausen_sf](#) (unsigned int __m, float __w)
- long double [clausen_sl](#) (unsigned int __m, long double __w)
- float [clausenf](#) (unsigned int __m, float __w)
- std::complex< float > [clausenf](#) (unsigned int __m, std::complex< float > __w)
- long double [clausenl](#) (unsigned int __m, long double __w)
- std::complex< long double > [clausenl](#) (unsigned int __m, std::complex< long double > __w)
- template<typename _Tk >
__gnu_cxx::__promote_num_t<_Tk > [comp_ellint_d](#) (_Tk __k)
- float [comp_ellint_df](#) (float __k)
- long double [comp_ellint_dl](#) (long double __k)
- float [comp_ellint_rf](#) (float __x, float __y)
- long double [comp_ellint_rf](#) (long double __x, long double __y)
- template<typename _Tx, typename _Ty >
__gnu_cxx::__promote_num_t<_Tx, _Ty > [comp_ellint_rf](#) (_Tx __x, _Ty __y)
- float [comp_ellint_rg](#) (float __x, float __y)
- long double [comp_ellint_rg](#) (long double __x, long double __y)
- template<typename _Tx, typename _Ty >
__gnu_cxx::__promote_num_t<_Tx, _Ty > [comp_ellint_rg](#) (_Tx __x, _Ty __y)
- template<typename _Tpa, typename _Tpc, typename _Tp >
__gnu_cxx::__promote_3<_Tpa, _Tpc, _Tp >::__type [conf_hyperg](#) (_Tpa __a, _Tpc __c, _Tp __x)
- template<typename _Tpc, typename _Tp >
__gnu_cxx::__promote_2<_Tpc, _Tp >::__type [conf_hyperg_lim](#) (_Tpc __c, _Tp __x)
- float [conf_hyperg_limf](#) (float __c, float __x)
- long double [conf_hyperg_liml](#) (long double __c, long double __x)
- float [conf_hypergf](#) (float __a, float __c, float __x)
- long double [conf_hypergl](#) (long double __a, long double __c, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [coshint](#) (_Tp __x)
- float [coshintf](#) (float __x)
- long double [coshintl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [cosint](#) (_Tp __x)
- float [cosintf](#) (float __x)
- long double [cosintl](#) (long double __x)
- template<typename _Tpnu, typename _Tp >
std::complex< __gnu_cxx::__promote_num_t<_Tpnu, _Tp > > [cyl_hankel_1](#) (_Tpnu __nu, _Tp __z)

- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > cyl_hankel_1 (std::complex< _Tpnu > __nu,`
`std::complex< _Tp > __x)`
- `std::complex< float > cyl_hankel_1f (float __nu, float __z)`
- `std::complex< float > cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > cyl_hankel_1l (long double __nu, long double __z)`
- `std::complex< long double > cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double >`
`__x)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > cyl_hankel_2 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > cyl_hankel_2 (std::complex< _Tpnu > __nu,`
`std::complex< _Tp > __x)`
- `std::complex< float > cyl_hankel_2f (float __nu, float __z)`
- `std::complex< float > cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > cyl_hankel_2l (long double __nu, long double __z)`
- `std::complex< long double > cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double >`
`__x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > dawson (_Tp __x)`
- `float dawsonf (float __x)`
- `long double dawsonl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > digamma (_Tp __z)`
- `float digammaf (float __z)`
- `long double digammal (long double __z)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > dilog (_Tp __x)`
- `float dilogf (float __x)`
- `long double dilogl (long double __x)`
- `template<typename _Tp >`
`_Tp dirichlet_beta (_Tp __s)`
- `float dirichlet_betaf (float __s)`
- `long double dirichlet_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 >`
`__gnu_cxx::__promote_num_t< _Tp > double_factorial (int __n)`
- `float double_factorialf (int __n)`
- `long double double_factoriall (int __n)`
- `template<typename _Tk, typename _Tp, typename _Ta, typename _Tb >`
`__gnu_cxx::__promote_num_t< _Tk, _Tp, _Ta, _Tb > ellint_cel (_Tk __k_c, _Tp __p, _Ta __a, _Tb __b)`
- `float ellint_celf (float __k_c, float __p, float __a, float __b)`
- `long double ellint_cell (long double __k_c, long double __p, long double __a, long double __b)`
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > ellint_d (_Tk __k, _Tphi __phi)`
- `float ellint_df (float __k, float __phi)`
- `long double ellint_dll (long double __k, long double __phi)`
- `template<typename _Tp, typename _Tk >`
`__gnu_cxx::__promote_num_t< _Tp, _Tk > ellint_el1 (_Tp __x, _Tk __k_c)`

- float [ellint_el1f](#) (float __x, float __k_c)
- long double [ellint_el1l](#) (long double __x, long double __k_c)
- template<typename _Tp, typename _Tk, typename _Ta, typename _Tb >
__gnu_cxx::__promote_num_t< _Tp, _Tk, _Ta, _Tb > [ellint_el2](#) (_Tp __x, _Tk __k_c, _Ta __a, _Tb __b)
- float [ellint_el2f](#) (float __x, float __k_c, float __a, float __b)
- long double [ellint_el2l](#) (long double __x, long double __k_c, long double __a, long double __b)
- template<typename _Tx, typename _Tk, typename _Tp >
__gnu_cxx::__promote_num_t< _Tx, _Tk, _Tp > [ellint_el3](#) (_Tx __x, _Tk __k_c, _Tp __p)
- float [ellint_el3f](#) (float __x, float __k_c, float __p)
- long double [ellint_el3l](#) (long double __x, long double __k_c, long double __p)
- template<typename _Tp, typename _Up >
__gnu_cxx::__promote_num_t< _Tp, _Up > [ellint_rc](#) (_Tp __x, _Up __y)
- float [ellint_rcf](#) (float __x, float __y)
- long double [ellint_rcl](#) (long double __x, long double __y)
- template<typename _Tp, typename _Up, typename _Vp >
__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > [ellint_rd](#) (_Tp __x, _Up __y, _Vp __z)
- float [ellint_rdf](#) (float __x, float __y, float __z)
- long double [ellint_rdl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > [ellint_rf](#) (_Tp __x, _Up __y, _Vp __z)
- float [ellint_rff](#) (float __x, float __y, float __z)
- long double [ellint_rfl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp > [ellint_rg](#) (_Tp __x, _Up __y, _Vp __z)
- float [ellint_rgf](#) (float __x, float __y, float __z)
- long double [ellint_rgl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp, typename _Wp >
__gnu_cxx::__promote_num_t< _Tp, _Up, _Vp, _Wp > [ellint_rj](#) (_Tp __x, _Up __y, _Vp __z, _Wp __p)
- float [ellint_rjf](#) (float __x, float __y, float __z, float __p)
- long double [ellint_rjl](#) (long double __x, long double __y, long double __z, long double __p)
- template<typename _Tp >
_Tp [ellnome](#) (_Tp __k)
- float [ellnomef](#) (float __k)
- long double [ellnomel](#) (long double __k)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [expint_e1](#) (_Tp __x)
- float [expint_e1f](#) (float __x)
- long double [expint_e1l](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [expint_en](#) (unsigned int __n, _Tp __x)
- float [expint_enf](#) (unsigned int __n, float __x)
- long double [expint_enl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [factorial](#) (unsigned int __n)
- float [factorialf](#) (unsigned int __n)
- long double [factoriall](#) (unsigned int __n)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [fresnel_c](#) (_Tp __x)
- float [fresnel_cf](#) (float __x)
- long double [fresnel_cl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [fresnel_s](#) (_Tp __x)

- float [fresnel_sf](#) (float __x)
- long double [fresnel_sl](#) (long double __x)
- template<typename _Tn, typename _Tp >
__gnu_cxx::__promote_num_t< _Tn, _Tp > [gamma_l](#) (_Tn __n, _Tp __x)
- float [gamma_lf](#) (float __n, float __x)
- long double [gamma_ll](#) (long double __n, long double __x)
- template<typename _Ta, typename _Tp >
__gnu_cxx::__promote_num_t< _Ta, _Tp > [gamma_p](#) (_Ta __a, _Tp __x)
- float [gamma_pf](#) (float __a, float __x)
- long double [gamma_pl](#) (long double __a, long double __x)
- template<typename _Ta, typename _Tp >
__gnu_cxx::__promote_num_t< _Ta, _Tp > [gamma_q](#) (_Ta __a, _Tp __x)
- float [gamma_qf](#) (float __a, float __x)
- long double [gamma_ql](#) (long double __a, long double __x)
- template<typename _Tn, typename _Tp >
__gnu_cxx::__promote_num_t< _Tn, _Tp > [gamma_u](#) (_Tn __n, _Tp __x)
- float [gamma_uf](#) (float __n, float __x)
- long double [gamma_ul](#) (long double __n, long double __x)
- template<typename _Talpha, typename _Tp >
__gnu_cxx::__promote_num_t< _Talpha, _Tp > [gegenbauer](#) (unsigned int __n, _Talpha __alpha, _Tp __x)
- float [gegenbauerf](#) (unsigned int __n, float __alpha, float __x)
- long double [gegenbauerl](#) (unsigned int __n, long double __alpha, long double __x)
- template<typename _Tk, typename _Tphi >
__gnu_cxx::__promote_num_t< _Tk, _Tphi > [heuman_lambda](#) (_Tk __k, _Tphi __phi)
- float [heuman_lambdaf](#) (float __k, float __phi)
- long double [heuman_lambdal](#) (long double __k, long double __phi)
- template<typename _Tp, typename _Up >
__gnu_cxx::__promote_num_t< _Tp, _Up > [hurwitz_zeta](#) (_Tp __s, _Up __a)
- template<typename _Tp, typename _Up >
std::complex< _Tp > [hurwitz_zeta](#) (_Tp __s, std::complex< _Up > __a)
- float [hurwitz_zetaf](#) (float __s, float __a)
- long double [hurwitz_zetal](#) (long double __s, long double __a)
- template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >
__gnu_cxx::__promote_4< _Tpa, _Tpb, _Tpc, _Tp >::type [hyperg](#) (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)
- float [hypergf](#) (float __a, float __b, float __c, float __x)
- long double [hypergl](#) (long double __a, long double __b, long double __c, long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > [ibeta](#) (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > [ibetac](#) (_Ta __a, _Tb __b, _Tp __x)
- float [ibetacf](#) (float __a, float __b, float __x)
- long double [ibetacL](#) (long double __a, long double __b, long double __x)
- float [ibetaf](#) (float __a, float __b, float __x)
- long double [ibetal](#) (long double __a, long double __b, long double __x)
- template<typename _Talpha, typename _Tbeta, typename _Tp >
__gnu_cxx::__promote_num_t< _Talpha, _Tbeta, _Tp > [jacobi](#) (unsigned __n, _Talpha __alpha, _Tbeta __beta, _Tp __x)
- template<typename _Kp, typename _Up >
__gnu_cxx::__promote_num_t< _Kp, _Up > [jacobi_cn](#) (_Kp __k, _Up __u)
- float [jacobi_cnf](#) (float __k, float __u)
- long double [jacobi_cnl](#) (long double __k, long double __u)

- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > jacobi_dn (_Kp __k, _Up __u)`
- `float jacobi_dnf (float __k, float __u)`
- `long double jacobi_dnl (long double __k, long double __u)`
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > jacobi_sn (_Kp __k, _Up __u)`
- `float jacobi_snf (float __k, float __u)`
- `long double jacobi_snl (long double __k, long double __u)`
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > jacobi_zeta (_Tk __k, _Tphi __phi)`
- `float jacobi_zetaf (float __k, float __phi)`
- `long double jacobi_zetal (long double __k, long double __phi)`
- `float jacobi_f (unsigned __n, float __alpha, float __beta, float __x)`
- `long double jacobi_l (unsigned __n, long double __alpha, long double __beta, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > lbincoef (unsigned int __n, unsigned int __k)`
- `float lbincoeff (unsigned int __n, unsigned int __k)`
- `long double lbincoeffl (unsigned int __n, unsigned int __k)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > ldouble_factorial (int __n)`
- `float ldouble_factorialf (int __n)`
- `long double ldouble_factoriall (int __n)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > legendre_q (unsigned int __n, _Tp __x)`
- `float legendre_qf (unsigned int __n, float __x)`
- `long double legendre_ql (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > lfactorial (unsigned int __n)`
- `float lfactorialf (unsigned int __n)`
- `long double lfactoriall (unsigned int __n)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > logint (_Tp __x)`
- `float logintf (float __x)`
- `long double logintl (long double __x)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > lpochhammer_l (_Tp __a, _Tn __n)`
- `float lpochhammer_lf (float __a, float __n)`
- `long double lpochhammer_ll (long double __a, long double __n)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > lpochhammer_u (_Tp __a, _Tn __n)`
- `float lpochhammer_uf (float __a, float __n)`
- `long double lpochhammer_ul (long double __a, long double __n)`
- `template<typename _Tph, typename _Tpa >`
`__gnu_cxx::__promote_num_t< _Tph, _Tpa > owens_t (_Tph __h, _Tpa __a)`
- `float owens_tf (float __h, float __a)`
- `long double owens_tl (long double __h, long double __a)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > pochhammer_l (_Tp __a, _Tn __n)`
- `float pochhammer_lf (float __a, float __n)`
- `long double pochhammer_ll (long double __a, long double __n)`
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > pochhammer_u (_Tp __a, _Tn __n)`

- float [pochhammer_uf](#) (float __a, float __n)
- long double [pochhammer_ul](#) (long double __a, long double __n)
- template<typename _Tp, typename _Wp >
__gnu_cxx::__promote_num_t<_Tp, _Wp > [polylog](#) (_Tp __s, _Wp __w)
- template<typename _Tp, typename _Wp >
std::complex< __gnu_cxx::__promote_num_t<_Tp, _Wp > > [polylog](#) (_Tp __s, std::complex<_Tp > __w)
- float [polylogf](#) (float __s, float __w)
- std::complex< float > [polylogf](#) (float __s, std::complex< float > __w)
- long double [polylogl](#) (long double __s, long double __w)
- std::complex< long double > [polylogl](#) (long double __s, std::complex< long double > __w)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [psi](#) (_Tp __x)
- float [psif](#) (float __x)
- long double [psil](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [radpoly](#) (unsigned int __n, unsigned int __m, _Tp __rho)
- float [radpolyf](#) (unsigned int __n, unsigned int __m, float __rho)
- long double [radpolyl](#) (unsigned int __n, unsigned int __m, long double __rho)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [sinc](#) (_Tp __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [sinc_pi](#) (_Tp __x)
- float [sinc_pif](#) (float __x)
- long double [sinc_pil](#) (long double __x)
- float [sincf](#) (float __x)
- long double [sincl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [sinhc](#) (_Tp __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [sinhc_pi](#) (_Tp __x)
- float [sinhc_pif](#) (float __x)
- long double [sinhc_pil](#) (long double __x)
- float [sinhcf](#) (float __x)
- long double [sinhcl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [sinhint](#) (_Tp __x)
- float [sinhintf](#) (float __x)
- long double [sinhintl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [sinint](#) (_Tp __x)
- float [sinintf](#) (float __x)
- long double [sinintl](#) (long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [sph_bessel_i](#) (unsigned int __n, _Tp __x)
- float [sph_bessel_if](#) (unsigned int __n, float __x)
- long double [sph_bessel_il](#) (unsigned int __n, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp > [sph_bessel_k](#) (unsigned int __n, _Tp __x)
- float [sph_bessel_kf](#) (unsigned int __n, float __x)
- long double [sph_bessel_kl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
std::complex< __gnu_cxx::__promote_num_t<_Tp > > [sph_hankel_1](#) (unsigned int __n, _Tp __z)

- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_1` (unsigned int __n, std::complex< _Tp > __x)
- `std::complex< float > sph_hankel_1f` (unsigned int __n, float __z)
- `std::complex< float > sph_hankel_1f` (unsigned int __n, std::complex< float > __x)
- `std::complex< long double > sph_hankel_1l` (unsigned int __n, long double __z)
- `std::complex< long double > sph_hankel_1l` (unsigned int __n, std::complex< long double > __x)
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_2` (unsigned int __n, _Tp __z)
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > sph_hankel_2` (unsigned int __n, std::complex< _Tp > __x)
- `std::complex< float > sph_hankel_2f` (unsigned int __n, float __z)
- `std::complex< float > sph_hankel_2f` (unsigned int __n, std::complex< float > __x)
- `std::complex< long double > sph_hankel_2l` (unsigned int __n, long double __z)
- `std::complex< long double > sph_hankel_2l` (unsigned int __n, std::complex< long double > __x)
- `template<typename _Ttheta , typename _Tphi >`
`std::complex< __gnu_cxx::__promote_num_t< _Ttheta , _Tphi > > sph_harmonic` (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)
- `std::complex< float > sph_harmonicf` (unsigned int __l, int __m, float __theta, float __phi)
- `std::complex< long double > sph_harmonicl` (unsigned int __l, int __m, long double __theta, long double __phi)
- `template<typename _Tpnu , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tpnu , _Tp > theta_1` (_Tpnu __nu, _Tp __x)
- `float theta_1f` (float __nu, float __x)
- `long double theta_1l` (long double __nu, long double __x)
- `template<typename _Tpnu , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tpnu , _Tp > theta_2` (_Tpnu __nu, _Tp __x)
- `float theta_2f` (float __nu, float __x)
- `long double theta_2l` (long double __nu, long double __x)
- `template<typename _Tpnu , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tpnu , _Tp > theta_3` (_Tpnu __nu, _Tp __x)
- `float theta_3f` (float __nu, float __x)
- `long double theta_3l` (long double __nu, long double __x)
- `template<typename _Tpnu , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tpnu , _Tp > theta_4` (_Tpnu __nu, _Tp __x)
- `float theta_4f` (float __nu, float __x)
- `long double theta_4l` (long double __nu, long double __x)
- `template<typename _Tp_k , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp_k , _Tp > theta_c` (_Tp_k __k, _Tp __x)
- `float theta_cf` (float __k, float __x)
- `long double theta_cl` (long double __k, long double __x)
- `template<typename _Tp_k , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp_k , _Tp > theta_d` (_Tp_k __k, _Tp __x)
- `float theta_df` (float __k, float __x)
- `long double theta_dl` (long double __k, long double __x)
- `template<typename _Tp_k , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp_k , _Tp > theta_n` (_Tp_k __k, _Tp __x)
- `float theta_nf` (float __k, float __x)
- `long double theta_nl` (long double __k, long double __x)
- `template<typename _Tp_k , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp_k , _Tp > theta_s` (_Tp_k __k, _Tp __x)
- `float theta_sf` (float __k, float __x)

- long double [theta_sl](#) (long double __k, long double __x)
- template<typename _Trho, typename _Tphi >
__gnu_cxx::__promote_num_t<_Trho, _Tphi > [zernike](#) (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)
- float [zernikef](#) (unsigned int __n, int __m, float __rho, float __phi)
- long double [zernikel](#) (unsigned int __n, int __m, long double __rho, long double __phi)

8.2 std Namespace Reference

Namespaces

- [__detail](#)

Functions

- template<typename _Tp >
__gnu_cxx::__promote<_Tp >::__type [assoc_laguerre](#) (unsigned int __n, unsigned int __m, _Tp __x)
- float [assoc_laguerref](#) (unsigned int __n, unsigned int __m, float __x)
- long double [assoc_laguerrel](#) (unsigned int __n, unsigned int __m, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote<_Tp >::__type [assoc_legendre](#) (unsigned int __l, unsigned int __m, _Tp __x)
- float [assoc_legendref](#) (unsigned int __l, unsigned int __m, float __x)
- long double [assoc_legendrel](#) (unsigned int __l, unsigned int __m, long double __x)
- template<typename _Tpa, typename _Tpb >
__gnu_cxx::__promote_2<_Tpa, _Tpb >::__type [beta](#) (_Tpa __a, _Tpb __b)
- float [betaf](#) (float __a, float __b)
- long double [betal](#) (long double __a, long double __b)
- template<typename _Tp >
__gnu_cxx::__promote<_Tp >::__type [comp_ellint_1](#) (_Tp __k)
- float [comp_ellint_1f](#) (float __k)
- long double [comp_ellint_1l](#) (long double __k)
- template<typename _Tp >
__gnu_cxx::__promote<_Tp >::__type [comp_ellint_2](#) (_Tp __k)
- float [comp_ellint_2f](#) (float __k)
- long double [comp_ellint_2l](#) (long double __k)
- template<typename _Tp, typename _Tpn >
__gnu_cxx::__promote_2<_Tp, _Tpn >::__type [comp_ellint_3](#) (_Tp __k, _Tpn __nu)
- float [comp_ellint_3f](#) (float __k, float __nu)
- *Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for float modulus k.*
- long double [comp_ellint_3l](#) (long double __k, long double __nu)
- *Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for long double modulus k.*
- template<typename _Tpnu, typename _Tp >
__gnu_cxx::__promote_2<_Tpnu, _Tp >::__type [cyl_bessel_i](#) (_Tpnu __nu, _Tp __x)
- float [cyl_bessel_if](#) (float __nu, float __x)
- long double [cyl_bessel_il](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
__gnu_cxx::__promote_2<_Tpnu, _Tp >::__type [cyl_bessel_j](#) (_Tpnu __nu, _Tp __x)
- float [cyl_bessel_jf](#) (float __nu, float __x)
- long double [cyl_bessel_jl](#) (long double __nu, long double __x)

- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type cyl_bessel_k (_Tpnu __nu, _Tp __x)`
- `float cyl_bessel_kf (float __nu, float __x)`
- `long double cyl_bessel_kl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::__type cyl_neumann (_Tpnu __nu, _Tp __x)`
- `float cyl_neumannf (float __nu, float __x)`
- `long double cyl_neumannl (long double __nu, long double __x)`
- `template<typename _Tp, typename _Tpp >`
`__gnu_cxx::__promote_2< _Tp, _Tpp >::__type ellint_1 (_Tp __k, _Tpp __phi)`
- `float ellint_1f (float __k, float __phi)`
- `long double ellint_1l (long double __k, long double __phi)`
- `template<typename _Tp, typename _Tpp >`
`__gnu_cxx::__promote_2< _Tp, _Tpp >::__type ellint_2 (_Tp __k, _Tpp __phi)`
- `float ellint_2f (float __k, float __phi)`
Return the incomplete elliptic integral of the second kind $E(k, \phi)$ for float argument.
- `long double ellint_2l (long double __k, long double __phi)`
Return the incomplete elliptic integral of the second kind $E(k, \phi)$.
- `template<typename _Tp, typename _Tpn, typename _Tpp >`
`__gnu_cxx::__promote_3< _Tp, _Tpn, _Tpp >::__type ellint_3 (_Tp __k, _Tpn __nu, _Tpp __phi)`
Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.
- `float ellint_3f (float __k, float __nu, float __phi)`
Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ for float argument.
- `long double ellint_3l (long double __k, long double __nu, long double __phi)`
Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type expint (_Tp __x)`
- `float expintf (float __x)`
- `long double expintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type hermite (unsigned int __n, _Tp __x)`
- `float hermitef (unsigned int __n, float __x)`
- `long double hermitel (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type laguerre (unsigned int __n, _Tp __x)`
- `float laguerref (unsigned int __n, float __x)`
- `long double laguerrel (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type legendre (unsigned int __l, _Tp __x)`
- `float legendref (unsigned int __l, float __x)`
- `long double legendrel (unsigned int __l, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type riemann_zeta (_Tp __s)`
- `float riemann_zetaf (float __s)`
- `long double riemann_zetal (long double __s)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type sph_bessel (unsigned int __n, _Tp __x)`
- `float sph_besself (unsigned int __n, float __x)`
- `long double sph_bessell (unsigned int __n, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`

- float [sph_legendref](#) (unsigned int __l, unsigned int __m, float __theta)
- long double [sph_legendrel](#) (unsigned int __l, unsigned int __m, long double __theta)
- template<typename _Tp >
__gnu_cxx::__promote< _Tp >::__type [sph_neumann](#) (unsigned int __n, _Tp __x)
- float [sph_neumannf](#) (unsigned int __n, float __x)
- long double [sph_neumannl](#) (unsigned int __n, long double __x)

8.3 std::__detail Namespace Reference

Classes

- struct [_Factorial_table](#)

Enumerations

- enum { [SININT](#), [COSINT](#) }

Functions

- template<typename _Tp >
void [__airy](#) (_Tp __z, _Tp &_Ai, _Tp &_Bi, _Tp &_Aip, _Tp &_Bip)
Compute the Airy functions $Ai(x)$ and $Bi(x)$ and their first derivatives $Ai'(x)$ and $Bi'(x)$ respectively.
- template<typename _Tp >
void [__airy](#) (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)
This function computes the Airy function $Ai(z)$ and its first derivative in the complex z-plane.
- template<typename _Tp >
std::complex< _Tp > [__airy_ai](#) (std::complex< _Tp > __z)
Return the complex Airy Ai function.
- template<typename _Tp >
void [__airy_arg](#) (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)
Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.
- template<typename _Tp >
void [__airy_asymp_absarg_ge_pio3](#) (std::complex< _Tp > __z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, int __sign=-1)
*This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(z)| < 2 * \pi/3$. For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from $abs(z)$.*
- template<typename _Tp >
void [__airy_asymp_absarg_lt_pio3](#) (std::complex< _Tp > __z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip)
This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(-z)| < \pi/3$. For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from $|z|$.
- template<typename _Tp >
void [__airy_bessel_i](#) (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_lp1d3, std::complex< _Tp > &_lm1d3, std::complex< _Tp > &_lp2d3, std::complex< _Tp > &_lm2d3)

- `template<typename _Tp >`
`void __airy_bessel_k (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Kp1d3, std::complex< _Tp > &_Kp2d3)`
Compute approximations to the modified Bessel functions of the second kind of orders 1/3 and 2/3 for moderate arguments.
- `template<typename _Tp >`
`std::complex< _Tp > __airy_bi (std::complex< _Tp > __z)`
Return the complex Airy Bi function.
- `template<typename _Tp >`
`void __airy_hyperg_rational (const std::complex< _Tp > &__z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)`
This function computes rational approximations to the hypergeometric functions related to the modified Bessel functions of orders $\nu = + - 1/3$ and $\nu + - 2/3$. That is, $A(z)/B(z)$, Where $A(z)$ and $B(z)$ are cubic polynomials with real coefficients, approximates

$$\frac{\Gamma(\nu + 1)}{(z/2)^n} I_\nu(z) = {}_0F_1(\nu + 1; z^2/4),$$
where the function on the right is a confluent hypergeometric limit function. For $|z| \leq 1/4$ and $|\arg(z)| \leq \pi/2$, the approximations are accurate to about 16 decimals.
- `template<typename _Tp >`
`_Tp __assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x)`
This routine returns the associated Laguerre polynomial of order n, degree m: $L_n^m(x)$.
- `template<typename _Tp >`
`_Tp __assoc_legendre_p (unsigned int __l, unsigned int __m, _Tp __x)`
Return the associated Legendre function by recursion on l and downward recursion on m.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __bernoulli (int __n)`
This returns Bernoulli number B_n .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __bernoulli_2n (int __n)`
This returns Bernoulli number B_n .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __bernoulli_series (unsigned int __n)`
This returns Bernoulli numbers from a table or by summation for larger values.
- `template<typename _Tp >`
`_Tp __beta (_Tp __a, _Tp __b)`
Return the beta function $B(a, b)$.
- `template<typename _Tp >`
`_Tp __beta_gamma (_Tp __a, _Tp __b)`
Return the beta function: $B(a, b)$.
- `template<typename _Tp >`
`_Tp __beta_inc (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`
`_Tp __beta_inc_cont_frac (_Tp __a, _Tp __b, _Tp __x)`
- `template<typename _Tp >`
`_Tp __beta_lgamma (_Tp __a, _Tp __b)`
Return the beta function $B(a, b)$ using the log gamma functions.
- `template<typename _Tp >`
`_Tp __beta_product (_Tp __a, _Tp __b)`
Return the beta function $B(x, y)$ using the product form.
- `template<typename _Tp >`
`_Tp __bincoef (unsigned int __n, unsigned int __k)`

Return the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)`
Return the binomial cumulative distribution function.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)`
Return the complementary binomial cumulative distribution function.
- `template<typename _Tp >`
`_Tp __bose_einstein (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_recur (unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1)`
- `template<typename _Tp >`
`_Tp __chebyshev_t (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_u (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_v (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_Tp __chebyshev_w (unsigned int __n, _Tp __x)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _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 >`
`_GLIBCXX14_CONSTEXPR _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 $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by series summation for positive argument.
- `template<typename _Tp >`
`std::complex< _Tp > __clamp_0_m2pi (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __clamp_pi (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __clausen (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp __clausen (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`
`_Tp __clausen_c (unsigned int __m, std::complex< _Tp > __w)`

- `template<typename _Tp >`
`_Tp __clausen_c` (unsigned int __m, _Tp __w)
- `template<typename _Tp >`
`_Tp __clausen_s` (unsigned int __m, std::complex< _Tp > __w)
- `template<typename _Tp >`
`_Tp __clausen_s` (unsigned int __m, _Tp __w)
- `template<typename _Tp >`
`_Tp __comp_ellint_1` (_Tp __k)
Return the complete elliptic integral of the first kind $K(k)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __comp_ellint_2` (_Tp __k)
Return the complete elliptic integral of the second kind $E(k)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __comp_ellint_3` (_Tp __k, _Tp __nu)
Return the complete elliptic integral of the third kind $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __comp_ellint_d` (_Tp __k)
- `template<typename _Tp >`
`_Tp __comp_ellint_rf` (_Tp __x, _Tp __y)
- `template<typename _Tp >`
`_Tp __comp_ellint_rg` (_Tp __x, _Tp __y)
- `template<typename _Tp >`
`_Tp __conf_hyperg` (_Tp __a, _Tp __c, _Tp __x)
Return the confluent hypergeometric function ${}_1F_1(a; c; x)$.
- `template<typename _Tp >`
`_Tp __conf_hyperg_lim` (_Tp __c, _Tp __x)
Return the confluent hypergeometric limit function ${}_0F_1(-; c; x)$.
- `template<typename _Tp >`
`_Tp __conf_hyperg_lim_series` (_Tp __c, _Tp __x)
This routine returns the confluent hypergeometric limit function by series expansion.
- `template<typename _Tp >`
`_Tp __conf_hyperg_luke` (_Tp __a, _Tp __c, _Tp __xin)
Return the hypergeometric function ${}_1F_1(a; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.
- `template<typename _Tp >`
`_Tp __conf_hyperg_series` (_Tp __a, _Tp __c, _Tp __x)
This routine returns the confluent hypergeometric function by series expansion.
- `template<typename _Tp >`
`_Tp __coshint` (const _Tp __x)
Return the hyperbolic cosine integral $li(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_bessel` (std::complex< _Tp > __nu, std::complex< _Tp > __z)
Return the complex cylindrical Bessel function.
- `template<typename _Tp >`
`_Tp __cyl_bessel_i` (_Tp __nu, _Tp __x)
Return the regular modified Bessel function of order ν : $I_\nu(x)$.
- `template<typename _Tp >`
`_Tp __cyl_bessel_ij_series` (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)
This routine returns the cylindrical Bessel functions of order ν : J_ν or I_ν by series expansion.

- `template<typename _Tp >`
`void __cyl_bessel_ik (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`
Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`void __cyl_bessel_ik_asymp (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`
This routine computes the asymptotic modified cylindrical Bessel and functions of order ν : $I_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.
- `template<typename _Tp >`
`void __cyl_bessel_ik_steeds (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`
Compute the modified Bessel functions $I_\nu(x)$ and $K_\nu(x)$ and their first derivatives $I'_\nu(x)$ and $K'_\nu(x)$ respectively. These four functions are computed together for numerical stability.
- `template<typename _Tp >`
`_Tp __cyl_bessel_j (_Tp __nu, _Tp __x)`
Return the Bessel function of order ν : $J_\nu(x)$.
- `template<typename _Tp >`
`void __cyl_bessel_jn (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
Return the cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`void __cyl_bessel_jn_asymp (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
This routine computes the asymptotic cylindrical Bessel and Neumann functions of order ν : $J_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.
- `template<typename _Tp >`
`void __cyl_bessel_jn_steeds (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
Compute the Bessel $J_\nu(x)$ and Neumann $N_\nu(x)$ functions and their first derivatives $J'_\nu(x)$ and $N'_\nu(x)$ respectively. These four functions are computed together for numerical stability.
- `template<typename _Tp >`
`_Tp __cyl_bessel_k (_Tp __nu, _Tp __x)`
Return the irregular modified Bessel function $K_\nu(x)$ of order ν .
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_hankel_1 (_Tp __nu, _Tp __x)`
Return the cylindrical Hankel function of the first kind $H_\nu^{(1)}(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_hankel_1 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Hankel function of the first kind.
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_hankel_2 (_Tp __nu, _Tp __x)`
Return the cylindrical Hankel function of the second kind $H_n^{(2)}u(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_hankel_2 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > __cyl_neumann (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Neumann function.
- `template<typename _Tp >`
`_Tp __cyl_neumann_n (_Tp __nu, _Tp __x)`
Return the Neumann function of order ν : $N_\nu(x)$.
- `template<typename _Tp >`
`_Tp __dawson (_Tp __x)`
Return the Dawson integral, $F(x)$, for real argument x .

- `template<typename _Tp >`
`_Tp __dawson_cont_frac (_Tp __x)`
Compute the Dawson integral using a sampling theorem representation.
- `template<typename _Tp >`
`_Tp __dawson_series (_Tp __x)`
Compute the Dawson integral using the series expansion.
- `template<typename _Tp >`
`void __debye_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)`
- `template<typename _Tp >`
`_Tp __dilog (_Tp __x)`
Compute the dilogarithm function $Li_2(x)$ by summation for $x \leq 1$.
- `template<typename _Tp >`
`_Tp __dirichlet_beta (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp __dirichlet_beta (_Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __dirichlet_eta (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp __dirichlet_eta (_Tp __w)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __double_factorial (int __n)`
Return the double factorial of the integer n .
- `template<typename _Tp >`
`_Tp __ellint_1 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_2 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the second kind $E(k, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)`
Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp __ellint_d (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`
`_Tp __ellint_el1 (_Tp __x, _Tp __k_c)`
- `template<typename _Tp >`
`_Tp __ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp __ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`
- `template<typename _Tp >`
`_Tp __ellint_rc (_Tp __x, _Tp __y)`
Return the Carlson elliptic function $R_C(x, y) = R_F(x, y, y)$ where $R_F(x, y, z)$ is the Carlson elliptic function of the first kind.
- `template<typename _Tp >`
`_Tp __ellint_rd (_Tp __x, _Tp __y, _Tp __z)`
Return the Carlson elliptic function of the second kind $R_D(x, y, z) = R_J(x, y, z, z)$ where $R_J(x, y, z, p)$ is the Carlson elliptic function of the third kind.
- `template<typename _Tp >`
`_Tp __ellint_rf (_Tp __x, _Tp __y, _Tp __z)`

Return the Carlson elliptic function $R_F(x, y, z)$ of the first kind.

- `template<typename _Tp >`
`_Tp __ellint_rg (_Tp __x, _Tp __y, _Tp __z)`

Return the symmetric Carlson elliptic function of the second kind $R_G(x, y, z)$.

- `template<typename _Tp >`
`_Tp __ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)`

Return the Carlson elliptic function $R_J(x, y, z, p)$ of the third kind.

- `template<typename _Tp >`
`_Tp __ellnome (_Tp __k)`
- `template<typename _Tp >`
`_Tp __ellnome_k (_Tp __k)`
- `template<typename _Tp >`
`_Tp __ellnome_series (_Tp __k)`
- `template<typename _Tp >`
`_Tp __expint (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$.

- `template<typename _Tp >`
`_Tp __expint (_Tp __x)`
- `template<typename _Tp >`
`_Tp __expint_asymp (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ for large argument.

- `template<typename _Tp >`
`_Tp __expint_E1 (_Tp __x)`
- `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)`
- `template<typename _Tp >`
`_Tp __expint_Ei_asymp (_Tp __x)`

Return the exponential integral $Ei(x)$ by asymptotic expansion.

- `template<typename _Tp >`
`_Tp __expint_Ei_series (_Tp __x)`

Return the exponential integral $Ei(x)$ by series summation.

- `template<typename _Tp >`
`_Tp __expint_En_cont_frac (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by continued fractions.

- `template<typename _Tp >`
`_Tp __expint_En_recursion (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.

- `template<typename _Tp >`
`_Tp __expint_En_series (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by series summation.

- `template<typename _Tp >`
`_Tp __expint_large_n (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ for large order.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __factorial (unsigned int __n)`
Return the factorial of the integer n .
- `template<typename _Tp >`
`_Tp __fermi_dirac (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`
Return the F -distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)`
Return the F -distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- `template<typename _Tp >`
`void __fock_airy (_Tp __x, std::complex< _Tp > &__w1, std::complex< _Tp > &__w2, std::complex< _Tp > &__w1p, std::complex< _Tp > &__w2p)`
Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w'_1(x)$ and $w'_2(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$

- `template<typename _Tp >`
`bool __fpequal (const _Tp &__a, const _Tp &__b)`
- `template<typename _Tp >`
`bool __fpimag (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`
`bool __fpimag (const _Tp)`
- `template<typename _Tp >`
`bool __fpreal (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`
`bool __fpreal (const _Tp)`
- `template<typename _Tp >`
`std::complex< _Tp > __fresnel (const _Tp __x)`
Return the Fresnel cosine and sine integrals as a complex number $\$f[C(x) + iS(x) \$f]$.
- `template<typename _Tp >`
`void __fresnel_cont_frac (const _Tp __ax, _Tp &__Cf, _Tp &__Sf)`
This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.
- `template<typename _Tp >`
`void __fresnel_series (const _Tp __ax, _Tp &__Cf, _Tp &__Sf)`
This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.
- `template<typename _Tp >`
`_Tp __gamma (_Tp __x)`
Return $\Gamma(x)$.
- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __gamma_cont_frac (_Tp __a, _Tp __x)`
- `template<typename _Tp >`
`_Tp __gamma_l (_Tp __a, _Tp __x)`

Return the lower incomplete gamma function. The lower incomplete gamma function is defined by

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`
`_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_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 >`
`std::pair<_Tp, _Tp> __gamma_series(_Tp __a, _Tp __x)`
- `template<typename _Tp >`
`void __gamma_temme(_Tp __mu, _Tp &__gam1, _Tp &__gam2, _Tp &__gampl, _Tp &__gammi)`

Compute the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ is $\mu = \nu - N$ and N is the nearest integer to ν . The values of $\Gamma(1+\mu)$ and $\Gamma(1-\mu)$ are returned as well.

- `template<typename _Tp >`
`_Tp __gamma_u(_Tp __a, _Tp __x)`

Return the upper incomplete gamma function. The lower incomplete gamma function is defined by

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`
`_Tp __gauss(_Tp __x)`
- `template<typename _Tp >`
`_Tp __gegenbauer_poly(unsigned int __n, _Tp __alpha, _Tp __x)`
- `template<typename _Tp >`
`void __hankel(std::complex<_Tp> __nu, std::complex<_Tp> __z, std::complex<_Tp> &__H1, std::complex<_Tp> &__H2, std::complex<_Tp> &__H1p, std::complex<_Tp> &__H2p)`

- template<typename _Tp >
 void [__hankel_debye](#) (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char & __aorb, int & __morn, std::complex< _Tp > & _H1, std::complex< _Tp > & _H2, std::complex< _Tp > & _H1p, std::complex< _Tp > & _H2p)
- template<typename _Tp >
 void [__hankel_params](#) (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > & __p, std::complex< _Tp > & __p2, std::complex< _Tp > & __nup2, std::complex< _Tp > & __num2, std::complex< _Tp > & __num1d3, std::complex< _Tp > & __num2d3, std::complex< _Tp > & __num4d3, std::complex< _Tp > & __zeta, std::complex< _Tp > & __zetaphf, std::complex< _Tp > & __zetamhf, std::complex< _Tp > & __zetam3hf, std::complex< _Tp > & __zetrat)

Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.
- template<typename _Tp >
 void [__hankel_uniform](#) (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > & _H1, std::complex< _Tp > & _H2, std::complex< _Tp > & _H1p, std::complex< _Tp > & _H2p)

This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.
- template<typename _Tp >
 void [__hankel_uniform_olver](#) (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > & __H1, std::complex< _Tp > & _H2, std::complex< _Tp > & _H1p, std::complex< _Tp > & _H2p)

Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order nu along with their derivatives.
- template<typename _Tp >
 void [__hankel_uniform_outer](#) (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > & __zhat, std::complex< _Tp > & __1dnsq, std::complex< _Tp > & __num1d3, std::complex< _Tp > & __num2d3, std::complex< _Tp > & __p, std::complex< _Tp > & __p2, std::complex< _Tp > & __etm3h, std::complex< _Tp > & __etrat, std::complex< _Tp > & __Aip, std::complex< _Tp > & __o4dp, std::complex< _Tp > & __Aim, std::complex< _Tp > & __o4dm, std::complex< _Tp > & __od2p, std::complex< _Tp > & __od0dp, std::complex< _Tp > & __od2m, std::complex< _Tp > & __od0dm)

Compute outer factors and associated functions of z and nu appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of z and nu returned by [hankel_uniform_outer](#) are available for use in computing further terms in the expansions.
- template<typename _Tp >
 void [__hankel_uniform_sum](#) (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > & __num2, std::complex< _Tp > & __zetam3hf, std::complex< _Tp > & __Aip, std::complex< _Tp > & __o4dp, std::complex< _Tp > & __Aim, std::complex< _Tp > & __o4dm, std::complex< _Tp > & __od2p, std::complex< _Tp > & __od0dp, std::complex< _Tp > & __od2m, std::complex< _Tp > & __od0dm, _Tp __eps, std::complex< _Tp > & __H1sum, std::complex< _Tp > & __H1psum, std::complex< _Tp > & __H2sum, std::complex< _Tp > & __H2psum)

Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to nterms (less than 5) to achieve relative error eps.
- template<typename _Tp >
 _Tp [__heuman_lambda](#) (_Tp __k, _Tp __phi)
- template<typename _Tp >
 _Tp [__hurwitz_zeta](#) (_Tp __s, _Tp __a)

Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.
- template<typename _Tp >
 std::complex< _Tp > [__hurwitz_zeta](#) (_Tp __s, std::complex< _Tp > __a)
- template<typename _Tp >
 _Tp [__hurwitz_zeta_euler_maclaurin](#) (_Tp __s, _Tp __a)

Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.

- `template<typename _Tp >`
`std::complex< _Tp > __hydrogen (const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp __Z, const _Tp __r, const _Tp __theta, const _Tp __phi)`
- `template<typename _Tp >`
`_Tp __hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`
Return the hypergeometric function ${}_2F_1(a, b; c; x)$.
- `template<typename _Tp >`
`_Tp __hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)`
Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.
- `template<typename _Tp >`
`_Tp __hyperg_reflect (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`
Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by the reflection formulae in Abramowitz & Stegun formula 15.3.6 for $d = c - a - b$ not integral and formula 15.3.11 for $d = c - a - b$ integral. This assumes $a, b, c \neq$ negative integer.
- `template<typename _Tp >`
`_Tp __hyperg_series (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`
Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by series expansion.
- `template<typename _Tp >`
`std::tuple< _Tp, _Tp, _Tp > __jacobi_sncndn (_Tp __k, _Tp __u)`
- `template<typename _Tp >`
`_Tp __jacobi_zeta (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`
`_Tp __laguerre (unsigned int __n, _Tp __x)`
This routine returns the Laguerre polynomial of order n : $L_n(x)$.
- `template<typename _Tp >`
`_Tp __legendre_q (unsigned int __l, _Tp __x)`
Return the Legendre function of the second kind by upward recursion on order l .
- `template<typename _Tp >`
`_Tp __log_bincoef (unsigned int __n, unsigned int __k)`
Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (_Tp __x)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_double_factorial (int __n)`
Return the logarithm of the double factorial of the integer n .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_factorial (unsigned int __n)`
Return the logarithm of the factorial of the integer n .
- `template<typename _Tp >`
`_Tp __log_gamma (_Tp __x)`
Return $\log(|\Gamma(x)|)$. This will return values even for $x < 0$. To recover the sign of $\Gamma(x)$ for any argument use `__log_gamma_sign`.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_bernoulli (_Tp __x)`
Return $\log(\Gamma(x))$ by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_lanczos (_Tp __x)`

Return $\log(\Gamma(x))$ by the Lanczos method. This method dominates all others on the positive axis I think.

- `template<typename _Tp >`
`_Tp __log_gamma_sign (_Tp __x)`

Return the sign of $\Gamma(x)$. At nonpositive integers zero is returned.

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __log_gamma_spouge (_Tp __z)`

Return $\Gamma(z)$ by the Spouge algorithm:

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[\sqrt{2\pi} \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

- `template<typename _Tp >`
`_Tp __log_pochhammer_l (_Tp __a, _Tp __n)`

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular, $\ln[(a)_n] = \ln n!$. Thus this function returns

$$\ln[(a)_n] = \Gamma(a+1) - \Gamma(a-n+1), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left\{ \begin{matrix} a \\ n \end{matrix} \right\}$$

, and others.

- `template<typename _Tp >`
`_Tp __log_pochhammer_u (_Tp __a, _Tp __n)`

Return the logarithm of the (upper) Pochhammer symbol or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(a)$$

Thus this function returns

$$\ln[(a)_n] = \Gamma(a+n) - \Gamma(a), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

- `template<typename _Tp >`
`_Tp __logint (const _Tp __x)`

Return the logarithmic integral $li(x)$.

- `template<typename _Tp >`
`_Tp __owens_t (_Tp __h, _Tp __a)`

- `template<typename _Tp >`
`_Tp __pochhammer_l(_Tp __a, _Tp __n)`
Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $(n)_n = n!$.

- `template<typename _Tp >`
`_Tp __pochhammer_u(_Tp __a, _Tp __n)`
Return the (upper) Pochhammer function or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a + k), (a)_0 = 1 = \Gamma(a + n) / \Gamma(a)$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\begin{bmatrix} a \\ n \end{bmatrix}$$

, and others.

- `template<typename _Tp >`
`_Tp __poly_hermite(unsigned int __n, _Tp __x)`
This routine returns the Hermite polynomial of order n : $H_n(x)$.
- `template<typename _Tp >`
`_Tp __poly_hermite_asymp(unsigned int __n, _Tp __x)`
This routine returns the Hermite polynomial of large order n : $H_n(x)$. We assume here that $x \geq 0$.
- `template<typename _Tp >`
`_Tp __poly_hermite_recursion(unsigned int __n, _Tp __x)`
This routine returns the Hermite polynomial of order n : $H_n(x)$ by recursion on n .
- `template<typename _Tp >`
`_Tp __poly_jacobi(unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)`
- `template<typename _Tpa, typename _Tp >`
`_Tp __poly_laguerre(unsigned int __n, _Tpa __alpha1, _Tp __x)`
This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^\alpha(x)$.
- `template<typename _Tpa, typename _Tp >`
`_Tp __poly_laguerre_hyperg(unsigned int __n, _Tpa __alpha1, _Tp __x)`
Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.
- `template<typename _Tpa, typename _Tp >`
`_Tp __poly_laguerre_large_n(unsigned int __n, _Tpa __alpha1, _Tp __x)`
This routine returns the associated Laguerre polynomial of order n , degree $\alpha > -1$ for large n . Abramowitz & Stegun, 13.5.21.
- `template<typename _Tpa, typename _Tp >`
`_Tp __poly_laguerre_recursion(unsigned int __n, _Tpa __alpha1, _Tp __x)`
This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^\alpha(x)$ by recursion.
- `template<typename _Tp >`
`_Tp __poly_legendre_p(unsigned int __l, _Tp __x)`
Return the Legendre polynomial by upward recursion on order l .
- `template<typename _Tp >`
`_Tp __poly_radial_jacobi(unsigned int __n, unsigned int __m, _Tp __rho)`
- `template<typename _Tp >`
`_Tp __polylog(_Tp __s, _Tp __x)`

- `template<typename _Tp >`
`std::complex< _Tp > __polylog (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp, typename ArgType >`
`__gnu_cxx::__promote_num_t< std::complex< _Tp >, ArgType > __polylog_exp (_Tp __s, ArgType __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_int_neg (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_int_neg (const int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_int_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_int_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_neg (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp, int __sigma>`
`std::complex< _Tp > __polylog_exp_neg_even (unsigned int __n, std::complex< _Tp > __w)`
- `template<typename _Tp, int __sigma>`
`std::complex< _Tp > __polylog_exp_neg_odd (unsigned int __n, std::complex< _Tp > __w)`
- `template<typename _PowTp, typename _Tp >`
`_Tp __polylog_exp_negative_real_part (_PowTp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_real_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_real_neg (_Tp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_real_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_real_pos (_Tp __s, _Tp __w)`
- `template<typename _Tp >`
`_Tp __psi (_Tp __x)`

Return the digamma function. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

For negative argument the reflection formula is used:

$$\psi(x) = \psi(1-x) - \pi \cot(\pi x)$$

- `template<typename _Tp >`
`_Tp __psi (unsigned int __n, _Tp __x)`

Return the polygamma function $\psi^{(n)}(x)$.

- template<typename _Tp >
_Tp [__psi_asymp](#) (_Tp __x)

Return the digamma function for large argument. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

- template<typename _Tp >
_Tp [__psi_series](#) (_Tp __x)

Return the digamma function by series expansion. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

- template<typename _Tp >
_Tp [__riemann_zeta](#) (_Tp __s)

Return the Riemann zeta function $\zeta(s)$.

- template<typename _Tp >
_Tp [__riemann_zeta_alt](#) (_Tp __s)

Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.

- template<typename _Tp >
_Tp [__riemann_zeta_euler_maclaurin](#) (_Tp __s)

Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.

- template<typename _Tp >
_Tp [__riemann_zeta_glob](#) (_Tp __s)

Evaluate the Riemann zeta function by series for all $s \neq 1$. Convergence is great until largish negative numbers. Then the convergence of the > 0 sum gets better.

- template<typename _Tp >
_Tp [__riemann_zeta_m_1](#) (_Tp __s)

Return the Riemann zeta function $\zeta(s) - 1$.

- template<typename _Tp >
_Tp [__riemann_zeta_m_1_sum](#) (_Tp __s)

Return the Riemann zeta function $\zeta(s) - 1$ by summation for $s > 1$. This is a small remainder for large s .

- template<typename _Tp >
_Tp [__riemann_zeta_product](#) (_Tp __s)

Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.

- template<typename _Tp >
_Tp [__riemann_zeta_sum](#) (_Tp __s)

Compute the Riemann zeta function $\zeta(s)$ by summation for $s > 1$.

- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp> [__sinc](#) (_Tp __a, _Tp __x)

Return the generalized sinus cardinal function

$$\text{sinc}_a(x) = \frac{\sin(\pi x/a)}{(\pi x/a)}$$

- template<typename _Tp >
__gnu_cxx::__promote_num_t<_Tp> [__sinc](#) (_Tp __x)

Return the normalized sinus cardinal function

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __sinc_pi (_Tp __x)`

Return the unnormalized sinus cardinal function

$$\operatorname{sinc}_{\pi}(x) = \frac{\sin(x)}{x}$$

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > __sincosint (_Tp __x)`

This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a pair.

- `template<typename _Tp >`
`void __sincosint_asymp (_Tp __t, _Tp &_Si, _Tp &_Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.

- `template<typename _Tp >`
`void __sincosint_cont_frac (_Tp __t, _Tp &_Si, _Tp &_Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.

- `template<typename _Tp >`
`void __sincosint_series (_Tp __t, _Tp &_Si, _Tp &_Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __sinhc (_Tp __a, _Tp __x)`

Return the generalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __sinhc (_Tp __x)`

Return the normalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}(x) = \frac{\sinh(\pi x)}{\pi x}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __sinhc_pi (_Tp __x)`

Return the unnormalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_{\pi}(x) = \frac{\sinh(x)}{x}$$

- `template<typename _Tp >`
`_Tp __sinhint (const _Tp __x)`

Return the hyperbolic sine integral $li(x)$.

- `template<typename _Tp >`
`_Tp __sph_bessel (unsigned int __n, _Tp __x)`

Return the spherical Bessel function $j_n(x)$ of order n and non-negative real argument x .

- `template<typename _Tp >`
`std::complex< _Tp > __sph_bessel (unsigned int __n, std::complex< _Tp > __z)`

Return the complex spherical Bessel function.

- `template<typename _Tp >`
`void __sph_bessel_ik (unsigned int __n, _Tp __x, _Tp &__i_n, _Tp &__k_n, _Tp &__ip_n, _Tp &__kp_n)`

Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i'_n(x)$ and $k'_n(x)$ respectively.

- `template<typename _Tp >`
`void __sph_bessel_jn (unsigned int __n, _Tp __x, _Tp &__j_n, _Tp &__n_n, _Tp &__jp_n, _Tp &__np_n)`
Compute the spherical Bessel $j_n(x)$ and Neumann $n_n(x)$ functions and their first derivatives $j'_n(x)$ and $n'_n(x)$ respectively.
- `template<typename _Tp >`
`void __sph_hankel (unsigned int __n, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2, std::complex< _Tp > &_H2p)`
Helper to compute complex spherical Hankel functions and their derivatives.
- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_1 (unsigned int __n, _Tp __x)`
Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_1 (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Hankel function of the first kind.
- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_2 (unsigned int __n, _Tp __x)`
Return the spherical Hankel function of the second kind $h_n^{(2)}(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > __sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > __sph_harmonic (unsigned int __l, int __m, _Tp __theta, _Tp __phi)`
Return the spherical harmonic function.
- `template<typename _Tp >`
`_Tp __sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`
Return the spherical associated Legendre function.
- `template<typename _Tp >`
`_Tp __sph_neumann (unsigned int __n, _Tp __x)`
Return the spherical Neumann function $n_n(x)$ of order n and non-negative real argument x .
- `template<typename _Tp >`
`std::complex< _Tp > __sph_neumann (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Neumann function.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __student_t_cdf (_Tp __t, unsigned int __nu)`
Return the Students T probability function.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __student_t_cdfc (_Tp __t, unsigned int __nu)`
Return the complement of the Students T probability function.
- `template<typename _Tp >`
`_Tp __theta_1 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_2 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_2_asymp (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_2_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_3 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_3_asymp (_Tp __nu, _Tp __x)`

- `template<typename _Tp >`
`_Tp __theta_3_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_4 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_c (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_d (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_n (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp __theta_s (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)`
- `template<typename _Tp >`
`_Tp __znorm1 (_Tp __x)`
- `template<typename _Tp >`
`_Tp __znorm2 (_Tp __x)`
- `template<typename _Tp = double>`
`_Tp evenzeta (unsigned int __k)`

Variables

- `constexpr size_t _Num_Euler_Maclaurin_zeta = 100`
- `constexpr _Factorial_table< long double > _S_double_factorial_table [301]`
- `constexpr long double _S_Euler_Maclaurin_zeta [_Num_Euler_Maclaurin_zeta]`
- `constexpr _Factorial_table< long double > _S_factorial_table [171]`
- `constexpr _Factorial_table< long double > _S_neg_double_factorial_table [999]`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_double_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_double_factorials< double > = 301`
- `template<>`
`constexpr std::size_t _S_num_double_factorials< float > = 57`
- `template<>`
`constexpr std::size_t _S_num_double_factorials< long double > = 301`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_factorials< double > = 171`
- `template<>`
`constexpr std::size_t _S_num_factorials< float > = 35`
- `template<>`
`constexpr std::size_t _S_num_factorials< long double > = 171`
- `template<typename _Tp >`
`constexpr std::size_t _S_num_neg_double_factorials = 0`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< double > = 150`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< float > = 27`
- `template<>`
`constexpr std::size_t _S_num_neg_double_factorials< long double > = 999`

- constexpr size_t [_S_num_zetam1](#) = 33
- constexpr long double [_S_zetam1](#) [[_S_num_zetam1](#)]

8.3.1 Enumeration Type Documentation

8.3.1.1 anonymous enum

Enumerator

SININT

COSINT

Definition at line 42 of file sf_trigint.tcc.

8.3.2 Function Documentation

8.3.2.1 `template<typename _Tp> void std::__detail::__airy (_Tp __z, _Tp &_Ai, _Tp &_Bi, _Tp &_Aip, _Tp &_Bip)`

Compute the Airy functions $Ai(x)$ and $Bi(x)$ and their first derivatives $Ai'(x)$ and $Bi'(x)$ respectively.

Parameters

<code>__z</code>	The argument of the Airy functions.
<code>_Ai</code>	The output Airy function of the first kind.
<code>_Bi</code>	The output Airy function of the second kind.
<code>_Aip</code>	The output derivative of the Airy function of the first kind.
<code>_Bip</code>	The output derivative of the Airy function of the second kind.

Definition at line 497 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ik()`, and `__cyl_bessel_jn()`.

8.3.2.2 `template<typename _Tp> void std::__detail::__airy (const std::complex<_Tp> &__z, _Tp __eps, std::complex<_Tp> &_Ai, std::complex<_Tp> &_Aip, std::complex<_Tp> &_Bi, std::complex<_Tp> &_Bip)`

This function computes the Airy function $Ai(z)$ and its first derivative in the complex z -plane.

The algorithm used exploits numerous representations of the Airy function and its derivative. The representations are recorded here for reference:

$$(1a) Ai(z) = \frac{\sqrt{z}}{3} (I_{-1/3}(\zeta) - I_{1/3}(\zeta))$$

$$(1b) Bi(z) = \sqrt{\frac{z}{3}} (I_{-1/3}(\zeta) + I_{1/3}(\zeta))$$

$$(2) Ai(z) = \frac{\sqrt{z/3}}{\pi} K_{1/3}(\zeta) = \frac{2^{2/3} 3^{-5/6}}{\sqrt{(\pi)}} z \exp(-\zeta) U(5/6; 5/3; 2\zeta)$$

$$(3a) Ai(-z) = \frac{\sqrt{z}}{3} (J_{-1/3}(\zeta) + J_{1/3}(\zeta))$$

$$(3b) Bi(-z) = \sqrt{\frac{z}{3}} (J_{-1/3}(\zeta) - J_{1/3}(\zeta))$$

$$(4a) Ai'(z) = \frac{z}{3}(I_{2/3}(\zeta) - I_{-2/3}(\zeta))$$

$$(4b) Bi'(z) = \frac{z}{\sqrt{3}}(I_{-2/3}(\zeta) + I_{2/3}(\zeta))$$

$$(5a) Ai'(z) = -\frac{z}{\pi\sqrt{3}}K(2/3)(\zeta) = -\frac{4^{2/3}3^{-7/6}}{\sqrt{\pi}}z^2 \exp(-\zeta)U(7/6; 7/3; 2\zeta)$$

$$(6a) Ai'(-z) = \frac{z}{3}(J_{2/3}(\zeta) - J_{-2/3}(\zeta)),$$

$$(6b) Bi'(-z) = \frac{z}{\sqrt{3}}(J_{-2/3}(\zeta) + J_{2/3}(\zeta)),$$

Where $\zeta = -\frac{2}{3}z^{3/2}$ and $U(a; b; z)$ is the confluent hypergeometric function defined in

See also

Stegun, I. A. and Abramowitz, M., Handbook of Mathematical Functions, Natl. Bureau of Standards, AMS 55, pp 504-515, 1964.

The asymptotic expansions derivable from these representations and Hankel's asymptotic expansions for the Bessel functions are used for large modulus of z . The implementation has taken advantage of the error bounds given in

See also

Olver, F. W. J., Error Bounds for Asymptotic Expansions, with an Application to Cylinder Functions of Large Argument, in Asymptotic Solutions of Differential Equations (Wilcox, Ed.), Wiley and Sons, pp 163-183, 1964

Olver, F. W. J., Asymptotics and Special Functions, Academic Press, pp 266-268, 1974.

For small modulus of z , a rational approximation is used. This approximant is derived from

Luke, Y. L., Mathematical Functions and their Approximations, Academic Press, pp 361-363, 1975.

The identities given below are for Bessel functions of the first kind in terms of modified Bessel functions of the first kind are also used with the rational approximant.

For moderate modulus of z , three techniques are used. Two use a backward recursion algorithm with (1), (3), (4), and (6). The third uses the confluent hypergeometric representations given by (2) and (5). The backward recursion algorithm generates values of the modified Bessel functions of the first kind of orders $+ \text{ or } - 1/3$ and $+ \text{ or } - 2/3$ for z in the right half plane. Values for the corresponding Bessel functions of the first kind are recovered via the identities

$$J_\nu(z) = \exp(\nu\pi i/2)I_\nu(z\exp(-\pi i/2)), 0 \leq \arg(z) \leq \pi/2$$

and

$$J_\nu(z) = \exp(-\nu\pi i/2)I_\nu(z\exp(\pi i/2)), -\pi/2 \leq \arg(z) < 0.$$

The particular backward recursion algorithm used is discussed in

See also

Olver, F. W. J., Numerical solution of second-order linear difference equations, NBS J. Res., Series B, VOL 71B, pp 111-129, 1967.

Olver, F. W. J. and Sookne, D. J., Note on backward recurrence algorithms, Math. Comp. Vol 26, No. 120, pp 941-947, Oct. 1972

Sookne, D. J., Bessel Functions I and J of Complex Argument and Integer Order, NBS J. Res., Series B, Vol 77B, Nos 3& 4, pp 111-113, July-December, 1973.

The following paper was also useful

See also

Cody, W. J., Preliminary report on software for the modified Bessel functions of the first kind, Applied Mathematics Division, Argonne National Laboratory, Tech. Memo. no. 357.

A backward recursion algorithm is also used to compute the confluent hypergeometric function. The recursion relations and a convergence theorem are given in

See also

Wimp, J., On the computation of Tricomi's psi function, Computing, Vol 13, pp 195-203, 1974.

Parameters

in	<code>__z</code>	The argument at which the Airy function and its derivative are computed.
in	<code>__eps</code>	Relative error required. Currently, eps is used only in the backward recursion algorithms.
out	<code>_Ai</code>	The value computed for $Ai(z)$.
out	<code>_Aip</code>	The value computed for $Ai'(z)$.
out	<code>_Bi</code>	The value computed for $Bi(z)$.
out	<code>_Bip</code>	The value computed for $Bi'(z)$.

Definition at line 1002 of file sf_airy.tcc.

References `__airy_asymp_absarg_ge_pio3()`, `__airy_asymp_absarg_lt_pio3()`, `__airy_bessel_i()`, `__airy_bessel_k()`, and `__airy_hyperg_rational()`.

Referenced by `__airy_ai()`, `__airy_bi()`, `__hankel_uniform_outer()`, and `__poly_hermite_asymp()`.

8.3.2.3 `template<typename _Tp> std::complex<_Tp> std::__detail::__airy_ai (std::complex<_Tp> __z)`

Return the complex Airy Ai function.

Definition at line 1139 of file sf_airy.tcc.

References `__airy()`.

8.3.2.4 `template<typename _Tp> void std::__detail::__airy_arg (std::complex<_Tp> __num2d3, std::complex<_Tp> __zeta, std::complex<_Tp> & __argp, std::complex<_Tp> & __argm)`

Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.

Parameters

in	<code>__num2d3</code>	$\nu^{(-2/3)}$ - output from <code>hankel_params</code> .
in	<code>__zeta</code>	ζ in the uniform asymptotic expansions - output from <code>hankel_params</code> .
out	<code>__argp</code>	$\exp(+2\pi i/3) * \nu^{(2/3)} * \zeta$.
out	<code>__argm</code>	$\exp(-2\pi i/3) * \nu^{(2/3)} * \zeta$.

Exceptions

<code>std::runtime_error</code> .

Definition at line 241 of file sf_hankel.tcc.

Referenced by `__hankel_uniform_outer()`.

8.3.2.5 `template<typename _Tp> void std::__detail::__airy_asymp_absarg_ge_pio3 (std::complex< _Tp> __z, std::complex< _Tp> & _Ai, std::complex< _Tp> & _Aip, int __sign = -1)`

This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(z)| < 2 * \pi/3$. For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from $abs(z)$.

Note that for speed and since this function is called by another, checks for valid arguments are not made.

See also

Digital Library of Mathematical Functions Sec. 9.7 Asymptotic Expansions <http://dlmf.nist.gov/9.7>

Parameters

in	<code>__z</code>	Complex input variable set equal to the value at which $Ai(z)$ and $Bi(z)$ and their derivative are evaluated. This function assumes $ z > 15$ and $ arg(z) < 2\pi/3$.
in, out	<code>_Ai</code>	The value computed for $Ai(z)$.
in, out	<code>_Aip</code>	The value computed for $Ai'(z)$.
in	<code>__sign</code>	The sign of the series terms and exponent. The default (-1) gives the Airy Ai functions for $ arg(z) < \pi$. The value +1 gives the Airy Bi functions for $ arg(z) < \pi/3$.

Definition at line 71 of file `sf_airy.tcc`.

Referenced by `__airy()`.

8.3.2.6 `template<typename _Tp> void std::__detail::__airy_asymp_absarg_lt_pio3 (std::complex< _Tp> __z, std::complex< _Tp> & _Ai, std::complex< _Tp> & _Aip)`

This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(-z)| < \pi/3$. For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from $|z|$.

Note that for speed and since this function is called by another, checks for valid arguments are not made. This function assumes $|z| > 15$ and $|arg(-z)| < \pi/3$.

Parameters

in	<code>__z</code>	The value at which the Airy function and its derivative are evaluated.
out	<code>_Ai</code>	The computed value of the Airy function $Ai(z)$.
out	<code>_Aip</code>	The computed value of the Airy function derivative $Ai'(z)$.

Definition at line 186 of file `sf_airy.tcc`.

Referenced by `__airy()`.

8.3.2.7 `template<typename _Tp> void std::__detail::__airy_bessel_i (const std::complex< _Tp> & __z, _Tp __eps, std::complex< _Tp> & _lp1d3, std::complex< _Tp> & _lm1d3, std::complex< _Tp> & _lp2d3, std::complex< _Tp> & _lm2d3)`

Compute the modified Bessel functions of the first kind of orders $+1/3$ and $+2/3$ needed to compute the Airy functions and their derivatives from their representation in terms of the modified Bessel functions. This function is only used for z less than two in modulus and in the closed right half plane. This stems from the fact that the values of the modified Bessel functions occurring in the representations of the Airy functions and their derivatives are almost equal for z large in the right half plane. This means that loss of significance occurs if these representations are used for z too large in magnitude. This algorithm is also not used for z too small, since a low order rational approximation can be used instead.

This routine is an implementation of a modified version of Miller's backward recurrence algorithm for computation by from the recurrence relation

$$I_{\nu-1} = (2\nu/z)I_{\nu} + I_{\nu+1}$$

satisfied by the modified Bessel functions of the first kind. the normalization relationship used is

$$\frac{z/2)^{\nu} e^z}{\Gamma(\nu+1)} = I_{\nu}(z) + 2 \sum_{k=1}^{\infty} \frac{(k+\nu)\Gamma(2\nu+k)}{k!\Gamma(1+2\nu)} I_{\nu+k}(z).$$

This modification of the algorithm is given in part in

Olver, F. W. J. and Sookne, D. J., Note on Backward Recurrence Algorithms, Math. of Comp., Vol. 26, no. 120, Oct. 1972.

And further elaborated for the Bessel functions in

Sookne, D. J., Bessel Functions I and J of Complex Argument and Integer Order, J. Res. NBS - Series B, Vol 77B, Nos. 3 & 4, July-December, 1973.

Insight was also gained from

Cody, W. J., Preliminary Report on Software for the Modified Bessel Functions of the First Kind, Argonne National Laboratory, Applied Mathematics Division, Tech. Memo. No. 357, August, 1980.

Cody implements the algorithm of Sookne for fractional order and nonnegative real argument. Like Cody, we do not change the convergence testing mechanism in any substantial way. However, we do trim the overhead by making the additional assumption that performing the convergence test for the functions of order 2/3 will suffice for order 1/3 as well. This assumption has not been established by rigorous analysis at this time. For speed the convergence tests are performed in the 1-norm instead of the usual Euclidean norm used in the complex plane using the inequality

$$|x| + |y| \leq \sqrt{2} \sqrt{x^2 + y^2}$$

Parameters

in	<code>__z</code>	The argument of the modified Bessel functions.
in	<code>__eps</code>	The maximum relative error required in the results.
out	<code>_lp1d3</code>	The value of $I_{\ell+1/3}(z)$.
out	<code>_lm1d3</code>	The value of $I_{\ell-1/3}(z)$.
out	<code>_lp2d3</code>	The value of $I_{\ell+2/3}(z)$.
out	<code>_lm2d3</code>	The value of $I_{\ell-2/3}(z)$.

Definition at line 390 of file sf_airy.tcc.

Referenced by `__airy()`.

8.3.2.8 `template<typename _Tp> void std::__detail::__airy_bessel_k (const std::complex<_Tp> & __z, _Tp __eps, std::complex<_Tp> & _Kp1d3, std::complex<_Tp> & _Kp2d3)`

Compute approximations to the modified Bessel functions of the second kind of orders 1/3 and 2/3 for moderate arguments.

This routine computes

$$E_{\nu}(z) = \exp z \sqrt{2z/\pi} K_{\nu}(z), \text{ for } \nu = 1/3 \text{ and } \nu = 2/3$$

using a rational approximation given in

Luke, Y. L., Mathematical functions and their approximations, Academic Press, pp 366-367, 1975.

Though the approximation converges in $|\arg(z)| \leq \pi$, The convergence weakens as $\text{abs}(\arg(z))$ increases. Also, in the case of real order between 0 and 1, convergence weakens as the order approaches 1. For these reasons, we only use this code for $|\arg(z)| \leq 3\pi/4$ and the convergence test is performed only for order 2/3.

The coding of this function is also influenced by the fact that it will only be used for about $2 \leq |z| \leq 15$. Hence, certain considerations of overflow, underflow, and loss of significance are unimportant for our purpose.

Parameters

in	<code>__z</code>	The value for which the quantity E_{nu} is to be computed. it is recommended that $abs(z)$ not be too small and that $ arg(z) \leq 3\pi/4$.
in	<code>__eps</code>	The maximum relative error allowable in the computed results. The relative error test is based on the comparison of successive iterates.
out	<code>_Kp1d3</code>	The value computed for $E_{+1/3}(z)$.
out	<code>_Kp2d3</code>	The value computed for $E_{+2/3}(z)$.

Note

In the worst case, say, $z = 2$ and $arg(z) = 3\pi/4$, 20 iterations should give 7 or 8 decimals of accuracy.

Definition at line 604 of file `sf_airy.tcc`.

Referenced by `__airy()`.

8.3.2.9 `template<typename _Tp> std::complex<_Tp> std::__detail::__airy_bi (std::complex<_Tp> __z)`

Return the complex Airy Bi function.

Definition at line 1152 of file `sf_airy.tcc`.

References `__airy()`.

8.3.2.10 `template<typename _Tp> void std::__detail::__airy_hyperg_rational (const std::complex<_Tp> & __z, std::complex<_Tp> & _Ai, std::complex<_Tp> & _Aip, std::complex<_Tp> & _Bi, std::complex<_Tp> & _Bip)`

This function computes rational approximations to the hypergeometric functions related to the modified Bessel functions of orders $\nu = + - 1/3$ and $\nu = -2/3$. That is, $A(z)/B(z)$, Where $A(z)$ and $B(z)$ are cubic polynomials with real coefficients, approximates

$$\frac{\Gamma(\nu + 1)}{(z/2)^{\nu}} I_{\nu}(z) = {}_0F_1(; \nu + 1; z^2/4),$$

where the function on the right is a confluent hypergeometric limit function. For $|z| \leq 1/4$ and $|arg(z)| \leq \pi/2$, the approximations are accurate to about 16 decimals.

For further details including the four term recurrence relation satisfied by the numerator and denominator polynomials in the higher order approximants, see

Luke, Y.L., Mathematical Functions and their Approximations, Academic Press, pp 361-363, 1975.

An asymptotic expression for the error is given as well as other useful expressions in the event one wants to extend this function to incorporate higher order approximants.

Note also that for speed and since this function is called by another, checks that are not absolutely necessary are not made.

Parameters

in	<code>__z</code>	The argument at which the hypergeometric given above is to be evaluated. Since the approximation is of fixed order, $ z $ must be small to ensure sufficient accuracy of the computed results.
----	------------------	--

out	<code>_Ai</code>	The Airy function $Ai(z)$.
out	<code>_Aip</code>	The Airy function derivative $Ai'(z)$.
out	<code>_Bi</code>	The Airy function $Bi(z)$.
out	<code>_Bip</code>	The Airy function derivative $Bi'(z)$.

Definition at line 787 of file sf_airy.tcc.

Referenced by `__airy()`.

8.3.2.11 `template<typename _Tp> _Tp std::__detail::__assoc_laguerre (unsigned int __n, unsigned int __m, _Tp __x)`

This routine returns the associated Laguerre polynomial of order n, degree m: $L_n^m(x)$.

The associated Laguerre polynomial is defined for integral $\alpha = m$ by:

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x)$$

where the Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})$$

Template Parameters

<code>_Tp</code>	The type of the parameter
------------------	---------------------------

Parameters

<code>__n</code>	The order
<code>__m</code>	The degree
<code>__x</code>	The argument

Returns

The value of the associated Laguerre polynomial of order n, degree m, and argument x.

Definition at line 301 of file sf_laguerre.tcc.

Referenced by `__hydrogen()`.

8.3.2.12 `template<typename _Tp> _Tp std::__detail::__assoc_legendre_p (unsigned int __l, unsigned int __m, _Tp __x)`

Return the associated Legendre function by recursion on l and downward recursion on m .

The associated Legendre function is derived from the Legendre function $P_l(x)$ by the Rodrigues formula:

$$P_l^m(x) = (1 - x^2)^{m/2} \frac{d^m}{dx^m} P_l(x)$$

Parameters

<code>__l</code>	The order of the associated Legendre function. $l \geq 0$.
<code>__m</code>	The order of the associated Legendre function. $m \leq l$.
<code>__x</code>	The argument of the associated Legendre function. $ x \leq 1$.

Definition at line 175 of file `sf_legendre.tcc`.

References `__poly_legendre_p()`.

8.3.2.13 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (int __n)`

This returns Bernoulli number B_n .

Parameters

<code>__n</code>	the order n of the Bernoulli number.
------------------	--------------------------------------

Returns

The Bernoulli number of order n.

Definition at line 1673 of file `sf_gamma.tcc`.

8.3.2.14 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (int __n)`

This returns Bernoulli number B_n .

Parameters

<code>__n</code>	the order n of the Bernoulli number.
------------------	--------------------------------------

Returns

The Bernoulli number of order n.

Definition at line 1685 of file `sf_gamma.tcc`.

8.3.2.15 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series (unsigned int __n)`

This returns Bernoulli numbers from a table or by summation for larger values.

Upward recursion is unstable.

Parameters

<code>__n</code>	the order n of the Bernoulli number.
------------------	--------------------------------------

Returns

The Bernoulli number of order n.

Definition at line 1608 of file `sf_gamma.tcc`.

8.3.2.16 `template<typename _Tp> _Tp std::__detail::__beta (_Tp __a, _Tp __b)`

Return the beta function $B(a, b)$.

The beta function is defined by

$$B(a, b) = \int_0^1 t^{a-1} (1-t)^{b-1} dt = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Parameters

<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 173 of file `sf_beta.tcc`.

References `__beta_lgamma()`.

Referenced by `__poly_jacobi()`, `__gnu_cxx::jacobi()`, `__gnu_cxx::jacobif()`, and `__gnu_cxx::jacobil()`.

8.3.2.17 `template<typename _Tp> _Tp std::__detail::__beta_gamma (_Tp __a, _Tp __b)`

Return the beta function: $B(a, b)$.

The beta function is defined by

$$B(a, b) = \int_0^1 t^{a-1} (1-t)^{b-1} dt = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Parameters

<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 75 of file `sf_beta.tcc`.

References `__gamma()`.

8.3.2.18 `template<typename _Tp> _Tp std::__detail::__beta_inc (_Tp __a, _Tp __b, _Tp __x)`

Return the regularized incomplete beta function, $I_x(a, b)$, of arguments `a`, `b`, and `x`.

The regularized incomplete beta function is defined by:

$$I_x(a, b) = \frac{B_x(a, b)}{B(a, b)}$$

where

$$B_x(a, b) = \int_0^x t^{a-1} (1-t)^{b-1} dt$$

is the non-regularized beta function and $B(a, b)$ is the usual beta function.

Parameters

<code>__a</code>	The first parameter
<code>__b</code>	The second parameter
<code>__x</code>	The argument

Definition at line 262 of file `sf_beta.tcc`.

References `__beta_inc_cont_frac()`.

Referenced by `__binomial_cdf()`, `__binomial_cdfc()`, `__fisher_f_cdf()`, `__fisher_f_cdfc()`, `__student_t_cdf()`, and `__student_t_cdfc()`.

8.3.2.19 `template<typename _Tp> _Tp std::__detail::__beta_inc_cont_frac (_Tp __a, _Tp __b, _Tp __x)`

Return the regularized incomplete beta function, $I_x(a, b)$, of arguments `a`, `b`, and `x`.

Parameters

<code>__a</code>	The first parameter
<code>__b</code>	The second parameter
<code>__x</code>	The argument

Definition at line 193 of file `sf_beta.tcc`.

Referenced by `__beta_inc()`.

8.3.2.20 `template<typename _Tp> _Tp std::__detail::__beta_lgamma (_Tp __a, _Tp __b)`

Return the beta function $B(a, b)$ using the log gamma functions.

The beta function is defined by

$$B(a, b) = \int_0^1 t^{a-1} (1-t)^{b-1} dt = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Parameters

<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 109 of file `sf_beta.tcc`.

References `__log_gamma()`.

Referenced by `__beta()`.

8.3.2.21 `template<typename _Tp> _Tp std::__detail::__beta_product (_Tp __a, _Tp __b)`

Return the beta function $B(x, y)$ using the product form.

The beta function is defined by

$$B(a, b) = \int_0^1 t^{a-1} (1-t)^{b-1} dt = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

Here, we employ the product form:

$$B(a, b) = \frac{a+b}{ab} \prod_{k=1}^{\infty} \frac{1 + (a+b)/k}{(1 + a/k)(1 + b/k)}$$

Parameters

<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 140 of file sf_beta.tcc.

8.3.2.22 `template<typename _Tp> _Tp std::__detail::__bincoef (unsigned int __n, unsigned int __k)`

Return the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

.

Parameters

<code>__n</code>	The first argument of the binomial coefficient.
<code>__k</code>	The second argument of the binomial coefficient.

Returns

The binomial coefficient.

Definition at line 1888 of file sf_gamma.tcc.

8.3.2.23 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdf (_Tp __p, unsigned int __n, unsigned int __k)`

Return the binomial cumulative distribution function.

The binomial cumulative distribution function is related to the incomplete beta function:

$$P(p|n, k) = I_p(k, n - k + 1)$$

Parameters

<code>__p</code>	
<code>__n</code>	
<code>__k</code>	

Definition at line 405 of file sf_beta.tcc.

References `__beta_inc()`.

8.3.2.24 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdfc (_Tp __p, unsigned int __n, unsigned int __k)`

Return the complementary binomial cumulative distribution function.

The binomial cumulative distribution function is related to the incomplete beta function:

$$Q(p|n, k) = I_{1-p}(n - k + 1, k)$$

Parameters

<code>__p</code>	
<code>__n</code>	
<code>__k</code>	

Definition at line 435 of file `sf_beta.tcc`.

References `__beta_inc()`.

8.3.2.25 `template<typename _Tp > _Tp std::__detail::__bose_einstein (_Tp __s, _Tp __x)`

Return the Bose-Einstein integral of real order s and real argument x .

See also

https://en.wikipedia.org/wiki/Clausen_function
<http://dlmf.nist.gov/25.12#iii>

Parameters

<code>__s</code>	The order $s \geq 0$.
<code>__x</code>	The real argument.

Returns

The real Fermi-Dirac cosine sum $G_s(x)$,

Definition at line 1401 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

8.3.2.26 `template<typename _Tp > _Tp std::__detail::__chebyshev_recur (unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1)`

Return a Chebyshev polynomial of non-negative order n and real argument x by the recursion

$$C_n(x) = 2xC_{n-1} - C_{n-2}$$

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

<code>__n</code>	The non-negative integral order
<code>__x</code>	The real argument $-1 \leq x \leq +1$
<code>_C0</code>	The value of the zeroth-order Chebyshev polynomial at x
<code>_C1</code>	The value of the first-order Chebyshev polynomial at x

Definition at line 57 of file sf_chebyshev.tcc.

Referenced by `__chebyshev_t()`, `__chebyshev_u()`, `__chebyshev_v()`, and `__chebyshev_w()`.

8.3.2.27 `template<typename _Tp> _Tp std::__detail::__chebyshev_t(unsigned int __n, _Tp __x)`

Return the Chebyshev polynomial of the first kind $T_n(x)$ of non-negative order n and real argument x .

The Chebyshev polynomial of the first kind is defined by:

$$T_n(x) = \cos(n\theta)$$

where $\theta = \arccos(x)$, $-1 \leq x \leq +1$.

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

<code>__n</code>	The non-negative integral order
<code>__x</code>	The real argument $-1 \leq x \leq +1$

Definition at line 85 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

8.3.2.28 `template<typename _Tp> _Tp std::__detail::__chebyshev_u(unsigned int __n, _Tp __x)`

Return the Chebyshev polynomial of the second kind $U_n(x)$ of non-negative order n and real argument x .

The Chebyshev polynomial of the second kind is defined by:

$$U_n(x) = \frac{\sin[(n+1)\theta]}{\sin(\theta)}$$

where $\theta = \arccos(x)$, $-1 \leq x \leq +1$.

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

<code>__n</code>	The non-negative integral order
<code>__x</code>	The real argument $-1 \leq x \leq +1$

Definition at line 114 of file sf_chebyshev.tcc.

References `__chebyshev_recur()`.

8.3.2.29 `template<typename _Tp> _Tp std::__detail::__chebyshev_v (unsigned int __n, _Tp __x)`

Return the Chebyshev polynomial of the third kind $V_n(x)$ of non-negative order n and real argument x .

The Chebyshev polynomial of the third kind is defined by:

$$V_n(x) = \frac{\cos \left[\left(n + \frac{1}{2} \right) \theta \right]}{\cos \left(\frac{\theta}{2} \right)}$$

where $\theta = \arccos(x)$, $-1 \leq x \leq +1$.

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

<code>__n</code>	The non-negative integral order
<code>__x</code>	The real argument $-1 \leq x \leq +1$

Definition at line 144 of file `sf_chebyshev.tcc`.

References `__chebyshev_recur()`.

8.3.2.30 `template<typename _Tp> _Tp std::__detail::__chebyshev_w (unsigned int __n, _Tp __x)`

Return the Chebyshev polynomial of the fourth kind $W_n(x)$ of non-negative order n and real argument x .

The Chebyshev polynomial of the fourth kind is defined by:

$$W_n(x) = \frac{\sin \left[\left(n + \frac{1}{2} \right) \theta \right]}{\sin \left(\frac{\theta}{2} \right)}$$

where $\theta = \arccos(x)$, $-1 \leq x \leq +1$.

Template Parameters

<code>_Tp</code>	The real type of the argument
------------------	-------------------------------

Parameters

<code>__n</code>	The non-negative integral order
<code>__x</code>	The real argument $-1 \leq x \leq +1$

Definition at line 174 of file `sf_chebyshev.tcc`.

References `__chebyshev_recur()`.

8.3.2.31 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _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 2544 of file `sf_gamma.tcc`.

References `__gamma_p()`.

8.3.2.32 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _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 2568 of file sf_gamma.tcc.

References `__gamma_q()`.

8.3.2.33 `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 162 of file sf_hypint.tcc.

References `__chshint_cont_frac()`, and `__chshint_series()`.

8.3.2.34 `template<typename _Tp> void std::__detail::__chshint_cont_frac (_Tp __t, _Tp & __Chi, _Tp & __Shi)`

This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by continued fraction for positive argument.

Definition at line 50 of file sf_hypint.tcc.

Referenced by `__chshint()`.

8.3.2.35 `template<typename _Tp> void std::__detail::__chshint_series (_Tp __t, _Tp & __Chi, _Tp & __Shi)`

This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by series summation for positive argument.

Definition at line 93 of file sf_hypint.tcc.

Referenced by `__chshint()`.

8.3.2.36 `template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_0_m2pi (std::complex<_Tp> __w)`

Definition at line 136 of file sf_polylog.tcc.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

8.3.2.37 `template<typename _Tp> std::complex<_Tp> std::__detail::__clamp_pi (std::complex<_Tp> __w)`

Definition at line 123 of file `sf_polylog.tcc`.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

8.3.2.38 `template<typename _Tp> std::complex<_Tp> std::__detail::__clausen (unsigned int __m, std::complex<_Tp> __w)`

Return Clausen's function of integer order m and complex argument w . The notation and connection to polylog is from Wikipedia

Parameters

<code>__m</code>	The non-negative integral order.
<code>__w</code>	The complex argument.

Returns

The complex Clausen function.

Definition at line 1230 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

8.3.2.39 `template<typename _Tp> _Tp std::__detail::__clausen (unsigned int __m, _Tp __w)`

Return Clausen's function of integer order m and real argument w . The notation and connection to polylog is from Wikipedia

Parameters

<code>__m</code>	The integer order $m \geq 1$.
<code>__w</code>	The real argument.

Returns

The Clausen function.

Definition at line 1254 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

8.3.2.40 `template<typename _Tp> _Tp std::__detail::__clausen_c (unsigned int __m, std::complex<_Tp> __w)`

Return Clausen's cosine sum Cl_m for positive integer order m and complex argument w .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

<code>__m</code>	The integer order $m \geq 1$.
<code>__w</code>	The real argument.

Returns

The Clausen cosine sum $Cl_m(w)$,

Definition at line 1329 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

8.3.2.41 `template<typename _Tp> _Tp std::__detail::__clausen_c(unsigned int __m, _Tp __w)`

Return Clausen's cosine sum Cl_m for positive integer order m and real argument w .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

<code>__m</code>	The integer order $m \geq 1$.
<code>__w</code>	The real argument.

Returns

The real Clausen cosine sum $Cl_m(w)$,

Definition at line 1354 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

8.3.2.42 `template<typename _Tp> _Tp std::__detail::__clausen_s(unsigned int __m, std::complex<_Tp> __w)`

Return Clausen's sine sum Sl_m for positive integer order m and complex argument w .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

<code>__m</code>	The integer order $m \geq 1$.
<code>__w</code>	The complex argument.

Returns

The Clausen sine sum $Sl_m(w)$,

Definition at line 1279 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

8.3.2.43 `template<typename _Tp> _Tp std::__detail::__clausen_s (unsigned int __m, _Tp __w)`

Return Clausen's sine sum Sl_m for positive integer order m and real argument w .

See also

https://en.wikipedia.org/wiki/Clausen_function

Parameters

<code>__m</code>	The integer order $m \geq 1$.
<code>__w</code>	The complex argument.

Returns

The Clausen sine sum $Sl_m(w)$,

Definition at line 1304 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

8.3.2.44 `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 565 of file `sf_ellint.tcc`.

References `__comp_ellint_rf()`.

Referenced by `__ellint_1()`, `__ellnome_k()`, `__jacobi_zeta()`, `__theta_c()`, `__theta_d()`, `__theta_n()`, and `__theta_s()`.

8.3.2.45 `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

$$E(k, \pi/2) = \int_0^{\pi/2} \sqrt{1 - k^2 \sin^2 \theta}$$

Parameters

<code>__k</code>	The modulus of the complete elliptic function.
------------------	--

Returns

The complete elliptic function of the second kind.

Definition at line 638 of file sf_ellint.tcc.

References `__ellint_rd()`, and `__ellint_rf()`.

Referenced by `__ellint_2()`.

8.3.2.46 `template<typename _Tp> _Tp std::__detail::__comp_ellint_3 (_Tp __k, _Tp __nu)`

Return the complete elliptic integral of the third kind $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$ using the Carlson formulation.

The complete elliptic integral of the third kind is defined as

$$\Pi(k, \nu) = \int_0^{\pi/2} \frac{d\theta}{(1 - \nu \sin^2 \theta) \sqrt{1 - k^2 \sin^2 \theta}}$$

Parameters

<code>__k</code>	The argument of the elliptic function.
<code>__nu</code>	The second argument of the elliptic function.

Returns

The complete elliptic function of the third kind.

Definition at line 727 of file sf_ellint.tcc.

References `__ellint_rf()`, and `__ellint_rj()`.

Referenced by `__ellint_3()`.

8.3.2.47 `template<typename _Tp> _Tp std::__detail::__comp_ellint_d (_Tp __k)`

Return the complete Legendre elliptic integral D.

Definition at line 832 of file sf_ellint.tcc.

References `__ellint_rd()`.

8.3.2.48 `template<typename _Tp> _Tp std::__detail::__comp_ellint_rf (_Tp __x, _Tp __y)`

Definition at line 235 of file sf_ellint.tcc.

Referenced by `__comp_ellint_1()`, and `__ellint_rf()`.

8.3.2.49 `template<typename _Tp> _Tp std::__detail::__comp_ellint_rg (_Tp __x, _Tp __y)`

Definition at line 346 of file sf_ellint.tcc.

Referenced by `__ellint_rg()`.

8.3.2.50 `template<typename _Tp> _Tp std::__detail::__conf_hyperg (_Tp __a, _Tp __c, _Tp __x)`

Return the confluent hypergeometric function ${}_1F_1(a; c; x)$.

Parameters

<code>__a</code>	The <i>numerator</i> parameter.
<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 281 of file `sf_hyperg.tcc`.

References `__conf_hyperg_luke()`, and `__conf_hyperg_series()`.

8.3.2.51 `template<typename _Tp> _Tp std::__detail::__conf_hyperg_lim (_Tp __c, _Tp __x)`

Return the confluent hypergeometric limit function ${}_0F_1(-; c; x)$.

Parameters

<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric limit function.

Returns

The confluent limit hypergeometric function.

Definition at line 109 of file `sf_hyperg.tcc`.

References `__conf_hyperg_lim_series()`.

8.3.2.52 `template<typename _Tp> _Tp std::__detail::__conf_hyperg_lim_series (_Tp __c, _Tp __x)`

This routine returns the confluent hypergeometric limit function by series expansion.

$${}_0F_1(-; c; x) = \Gamma(c) \sum_{n=0}^{\infty} \frac{1}{\Gamma(c+n)} \frac{x^n}{n!}$$

If a and b are integers and $a < 0$ and either $b > 0$ or $b < a$ then the series is a polynomial with a finite number of terms.

Parameters

<code>__c</code>	The "denominator" parameter.
<code>__x</code>	The argument of the confluent hypergeometric limit function.

Returns

The confluent hypergeometric limit function.

Definition at line 76 of file `sf_hyperg.tcc`.

Referenced by `__conf_hyperg_lim()`.

8.3.2.53 `template<typename _Tp> _Tp std::__detail::__conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)`

Return the hypergeometric function ${}_1F_1(a; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.

Like the case of the $2F_1$ rational approximations, these are probably guaranteed to converge for $x < 0$, barring gross numerical instability in the pre-asymptotic regime.

Definition at line 176 of file `sf_hyperg.tcc`.

Referenced by `__conf_hyperg()`.

8.3.2.54 `template<typename _Tp> _Tp std::__detail::__conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)`

This routine returns the confluent hypergeometric function by series expansion.

$${}_1F_1(a; c; x) = \frac{\Gamma(c)}{\Gamma(a)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

Parameters

<code>__a</code>	The "numerator" parameter.
<code>__c</code>	The "denominator" parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 141 of file `sf_hyperg.tcc`.

Referenced by `__conf_hyperg()`.

8.3.2.55 `template<typename _Tp> _Tp std::__detail::__coshint (const _Tp __x)`

Return the hyperbolic cosine integral $li(x)$.

The hyperbolic cosine integral is given by

$$Chi(x) = (Ei(x) - E_1(x))/2$$

Parameters

<code>__x</code>	The argument of the hyperbolic cosine integral function.
------------------	--

Returns

The hyperbolic cosine integral.

Definition at line 558 of file `sf_expint.tcc`.

References `__expint_E1()`, and `__expint_Ei()`.

8.3.2.56 `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_bessel (std::complex<_Tp> __nu, std::complex<_Tp> __z)`

Return the complex cylindrical Bessel function.

Parameters

in	<code>__nu</code>	The order for which the cylindrical Bessel function is evaluated.
in	<code>__z</code>	The argument at which the cylindrical Bessel function is evaluated.

Returns

The complex cylindrical Bessel function.

Definition at line 1222 of file `sf_hankel.tcc`.

References `__hankel()`.

8.3.2.57 `template<typename _Tp> _Tp std::__detail::__cyl_bessel_i (_Tp __nu, _Tp __x)`

Return the regular modified Bessel function of order ν : $I_\nu(x)$.

The regular modified cylindrical Bessel function is:

$$I_\nu(x) = \sum_{k=0}^{\infty} \frac{(x/2)^{\nu+2k}}{k! \Gamma(\nu + k + 1)}$$

Parameters

<code>__nu</code>	The order of the regular modified Bessel function.
<code>__x</code>	The argument of the regular modified Bessel function.

Returns

The output regular modified Bessel function.

Definition at line 386 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_ik()`.

8.3.2.58 `template<typename _Tp> _Tp std::__detail::__cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)`

This routine returns the cylindrical Bessel functions of order ν : J_ν or I_ν by series expansion.

The modified cylindrical Bessel function is:

$$Z_\nu(x) = \sum_{k=0}^{\infty} \frac{\sigma^k (x/2)^{\nu+2k}}{k! \Gamma(\nu + k + 1)}$$

where $\sigma = +1$ or -1 for $Z = I$ or J respectively.

See Abramowitz & Stegun, 9.1.10 Abramowitz & Stegun, 9.6.7 (1) Handbook of Mathematical Functions, ed. Milton Abramowitz and Irene A. Stegun, Dover Publications, Equation 9.1.10 p. 360 and Equation 9.6.10 p. 375

Parameters

<code>__nu</code>	The order of the Bessel function.
<code>__x</code>	The argument of the Bessel function.
<code>__sgn</code>	The sign of the alternate terms -1 for the Bessel function of the first kind. +1 for the modified Bessel function of the first kind.
<code>__max_iter</code>	The maximum number of iterations for sum.

Returns

The output Bessel function.

Definition at line 413 of file `sf_bessel.tcc`.

References `__log_gamma()`.

Referenced by `__cyl_bessel_i()`, and `__cyl_bessel_j()`.

8.3.2.59 `template<typename _Tp> void std::__detail::__cyl_bessel_ik (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`

Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.

Parameters

<code>__nu</code>	The order of the Bessel functions.
<code>__x</code>	The argument of the Bessel functions.
<code>_Inu</code>	The output regular modified Bessel function.
<code>_Knu</code>	The output irregular modified Bessel function.
<code>_Ipnu</code>	The output derivative of the regular modified Bessel function.
<code>_Kpnu</code>	The output derivative of the irregular modified Bessel function.

Definition at line 316 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik_asymp()`, and `__cyl_bessel_ik_steel()`.

Referenced by `__airy()`, `__cyl_bessel_i()`, `__cyl_bessel_k()`, and `__sph_bessel_ik()`.

8.3.2.60 `template<typename _Tp> void std::__detail::__cyl_bessel_ik_asymp (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`

This routine computes the asymptotic modified cylindrical Bessel and functions of order ν : $I_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.

References: (1) Handbook of Mathematical Functions, ed. Milton Abramowitz and Irene A. Stegun, Dover Publications, Section 9 p. 364, Equations 9.2.5-9.2.10

Parameters

<code>__nu</code>	The order of the Bessel functions.
<code>__x</code>	The argument of the Bessel functions.
<code>_Inu</code>	The output regular modified Bessel function.
<code>_Knu</code>	The output irregular modified Bessel function.
<code>_Ipnu</code>	The output derivative of the regular modified Bessel function.

<code>__Kpnu</code>	The output derivative of the irregular modified Bessel function.
---------------------	--

Definition at line 81 of file `sf_mod_bessel.tcc`.

Referenced by `__cyl_bessel_ik()`, and `__cyl_bessel_ik_stepped()`.

8.3.2.61 `template<typename _Tp> void std::__detail::__cyl_bessel_ik_stepped (_Tp __nu, _Tp __x, _Tp & __Inu, _Tp & __Knu, _Tp & __Ipnu, _Tp & __Kpnu)`

Compute the modified Bessel functions $I_\nu(x)$ and $K_\nu(x)$ and their first derivatives $I'_\nu(x)$ and $K'_\nu(x)$ respectively. These four functions are computed together for numerical stability.

Parameters

<code>__nu</code>	The order of the Bessel functions.
<code>__x</code>	The argument of the Bessel functions.
<code>__Inu</code>	The output regular modified Bessel function.
<code>__Knu</code>	The output irregular modified Bessel function.
<code>__Ipnu</code>	The output derivative of the regular modified Bessel function.
<code>__Kpnu</code>	The output derivative of the irregular modified Bessel function.

Definition at line 152 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_ik()`.

8.3.2.62 `template<typename _Tp> _Tp std::__detail::__cyl_bessel_j (_Tp __nu, _Tp __x)`

Return the Bessel function of order ν : $J_\nu(x)$.

The cylindrical Bessel function is:

$$J_\nu(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{\nu+2k}}{k! \Gamma(\nu+k+1)}$$

Parameters

<code>__nu</code>	The order of the Bessel function.
<code>__x</code>	The argument of the Bessel function.

Returns

The output Bessel function.

Definition at line 523 of file `sf_bessel.tcc`.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_jn()`.

8.3.2.63 `template<typename _Tp> void std::__detail::__cyl_bessel_jn (_Tp __nu, _Tp __x, _Tp & __Jnu, _Tp & __Nnu, _Tp & __Jpnu, _Tp & __Npnu)`

Return the cylindrical Bessel functions and their derivatives of order ν by various means.

Definition at line 442 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn_asymp()`, and `__cyl_bessel_jn_stepped()`.

Referenced by `__airy()`, `__cyl_bessel_jn()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann_n()`, and `__sph_cyl_bessel_jn()`.

8.3.2.64 `template<typename _Tp> void std::__detail::__cyl_bessel_jn_asymp (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`

This routine computes the asymptotic cylindrical Bessel and Neumann functions of order ν : $J_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.

References: (1) Handbook of Mathematical Functions, ed. Milton Abramowitz and Irene A. Stegun, Dover Publications, Section 9 p. 364, Equations 9.2.5-9.2.10

Parameters

	<code>__nu</code>	The order of the Bessel functions.
	<code>__x</code>	The argument of the Bessel functions.
out	<code>_Jnu</code>	The Bessel function of the first kind.
out	<code>_Nnu</code>	The Neumann function (Bessel function of the second kind).
out	<code>_Jpnu</code>	The Bessel function of the first kind.
out	<code>_Npnu</code>	The Neumann function (Bessel function of the second kind).

Definition at line 79 of file `sf_bessel.tcc`.

Referenced by `__cyl_bessel_jn()`, and `__cyl_bessel_jn_steel()`.

8.3.2.65 `template<typename _Tp> void std::__detail::__cyl_bessel_jn_steel (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`

Compute the Bessel $J_\nu(x)$ and Neumann $N_\nu(x)$ functions and their first derivatives $J'_\nu(x)$ and $N'_\nu(x)$ respectively. These four functions are computed together for numerical stability.

Parameters

	<code>__nu</code>	The order of the Bessel functions.
	<code>__x</code>	The argument of the Bessel functions.
out	<code>_Jnu</code>	The output Bessel function of the first kind.
out	<code>_Nnu</code>	The output Neumann function (Bessel function of the second kind).
out	<code>_Jpnu</code>	The output derivative of the Bessel function of the first kind.
out	<code>_Npnu</code>	The output derivative of the Neumann function.

Definition at line 197 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_jn()`.

8.3.2.66 `template<typename _Tp> _Tp std::__detail::__cyl_bessel_k (_Tp __nu, _Tp __x)`

Return the irregular modified Bessel function $K_\nu(x)$ of order 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 424 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`.

8.3.2.67 `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_1 (_Tp __nu, _Tp __x)`

Return the cylindrical Hankel function of the first kind $H_\nu^{(1)}(x)$.

The cylindrical Hankel function of the first kind is defined by:

$$H_\nu^{(1)}(x) = J_\nu(x) + iN_\nu(x)$$

Parameters

<code>__nu</code>	The order of the spherical Neumann function.
<code>__x</code>	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Definition at line 588 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

8.3.2.68 `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_1 (std::complex<_Tp> __nu, std::complex<_Tp> __z)`

Return the complex cylindrical Hankel function of the first kind.

Parameters

<code>in</code>	<code>__nu</code>	The order for which the cylindrical Hankel function of the first kind is evaluated.
<code>in</code>	<code>__z</code>	The argument at which the cylindrical Hankel function of the first kind is evaluated.

Returns

The complex cylindrical Hankel function of the first kind.

Definition at line 1190 of file `sf_hankel.tcc`.

References `__hankel()`.

8.3.2.69 `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_2 (_Tp __nu, _Tp __x)`

Return the cylindrical Hankel function of the second kind $H_n^{(2)}u(x)$.

The cylindrical Hankel function of the second kind is defined by:

$$H_{\nu}^{(2)}(x) = J_{\nu}(x) - iN_{\nu}(x)$$

Parameters

<code>__nu</code>	The order of the spherical Neumann function.
<code>__x</code>	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Definition at line 624 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

8.3.2.70 `template<typename _Tp > std::complex<_Tp> std::__detail::__cyl_hankel_2 (std::complex<_Tp > __nu,
std::complex<_Tp > __z)`

Return the complex cylindrical Hankel function of the second kind.

Parameters

<code>in</code>	<code>__nu</code>	The order for which the cylindrical Hankel function of the second kind is evaluated.
<code>in</code>	<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 1206 of file `sf_hankel.tcc`.

References `__hankel()`.

8.3.2.71 `template<typename _Tp > std::complex<_Tp> std::__detail::__cyl_neumann (std::complex<_Tp > __nu,
std::complex<_Tp > __z)`

Return the complex cylindrical Neumann function.

Parameters

<code>in</code>	<code>__nu</code>	The order for which the cylindrical Neumann function is evaluated.
<code>in</code>	<code>__z</code>	The argument at which the cylindrical Neumann function is evaluated.

Returns

The complex cylindrical Neumann function.

Definition at line 1238 of file `sf_hankel.tcc`.

References `__hankel()`.

8.3.2.72 `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 558 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

8.3.2.73 `template<typename _Tp> _Tp std::__detail::__dawson (_Tp __x)`

Return the Dawson integral, $F(x)$, for real argument x .

The Dawson integral is defined by:

$$F(x) = e^{-x^2} \int_0^x e^{y^2} dy$$

and it's derivative is:

$$F'(x) = 1 - 2xF(x)$$

Parameters

<code>__x</code>	The argument $-\infty < x < \infty$.
------------------	---------------------------------------

Definition at line 233 of file `sf_dawson.tcc`.

References `__dawson_cont_frac()`, and `__dawson_series()`.

8.3.2.74 `template<typename _Tp> _Tp std::__detail::__dawson_cont_frac (_Tp __x)`

Compute the Dawson integral using a sampling theorem representation.

Todo this needs some compile-time construction!

Definition at line 71 of file `sf_dawson.tcc`.

Referenced by `__dawson()`.

8.3.2.75 `template<typename _Tp> _Tp std::__detail::__dawson_series (_Tp __x)`

Compute the Dawson integral using the series expansion.

Definition at line 47 of file `sf_dawson.tcc`.

Referenced by `__dawson()`.

8.3.2.76 `template<typename _Tp> void std::__detail::__debye_region (std::complex< _Tp> __alpha, int & __indexr, char & __aorb)`

Compute the Debye region in the complex plane.

Definition at line 54 of file `sf_hankel.tcc`.

Referenced by `__hankel()`.

8.3.2.77 `template<typename _Tp> _Tp std::__detail::__dilog (_Tp __x)`

Compute the dilogarithm function $Li_2(x)$ by summation for $x \leq 1$.

The Riemann zeta function is defined by:

$$Li_2(x) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For $|x|$ near 1 use the reflection formulae:

$$Li_2(-x) + Li_2(1-x) = \frac{\pi^2}{6} - \ln(x) \ln(1-x)$$

$$Li_2(-x) - Li_2(1-x) - \frac{1}{2}Li_2(1-x^2) = -\frac{\pi^2}{12} - \ln(x) \ln(1-x)$$

For $x < 1$ use the reflection formula:

$$Li_2(1-x) - Li_2\left(1 - \frac{1}{1-x}\right) - \frac{1}{2}(\ln(x))^2$$

Definition at line 194 of file sf_zeta.tcc.

8.3.2.78 `template<typename _Tp> _Tp std::__detail::__dirichlet_beta (std::complex<_Tp> __w)`

Return the Dirichlet beta function. Currently, w must be real (complex type but negligible imaginary part.) Otherwise std::domain_error is thrown.

Parameters

<code>__w</code>	The complex (but on-real-axis) argument.
------------------	--

Returns

The Dirichlet Beta function of real argument.

Exceptions

<code>std::domain_error</code>	if the argument has a significant imaginary part.
--------------------------------	---

Definition at line 1192 of file sf_polylog.tcc.

References `__fpequal()`, and `__polylog()`.

8.3.2.79 `template<typename _Tp> _Tp std::__detail::__dirichlet_beta (_Tp __w)`

Return the Dirichlet beta function for real argument.

Parameters

<code>__w</code>	The real argument.
------------------	--------------------

Returns

The Dirichlet Beta function of real argument.

Definition at line 1211 of file sf_polylog.tcc.

References `__polylog()`.

8.3.2.80 `template<typename _Tp > std::complex<_Tp> std::__detail::__dirichlet_eta (std::complex<_Tp > __w)`

Return the Dirichlet eta function. Currently, `w` must be real (complex type but negligible imaginary part.) Otherwise `std::domain_error` is thrown.

Parameters

<code>__w</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 1155 of file sf_polylog.tcc.

References `__fpequal()`, and `__polylog()`.

8.3.2.81 `template<typename _Tp> _Tp std::__detail::__dirichlet_eta (_Tp __w)`

Return the Dirichlet eta function for real argument.

Parameters

<code>__w</code>	The real argument.
------------------	--------------------

Returns

The Dirichlet eta function.

Definition at line 1173 of file sf_polylog.tcc.

References `__polylog()`.

8.3.2.82 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__double_factorial (int __n)`

Return the double factorial of the integer n.

The double factorial is defined for integral n by:

$$n!! = 135\dots(n-2)n, \text{ odd } n!! = 246\dots(n-2)n, \text{ even } n!! = 10!! = 1$$

The double factorial is defined for odd negative integers in the obvious way:

$$(-2m-1)!! = 1/(1(-1)(-3)\dots(-2m+1)(-2m-1)) = \frac{(-1)^m}{(2m-1)!!}$$

for $n = -2m - 1$.

Definition at line 2480 of file sf_gamma.tcc.

References `__factorial()`, `__log_double_factorial()`, `_S_double_factorial_table`, and `_S_neg_double_factorial_table`.

8.3.2.83 `template<typename _Tp> _Tp std::__detail::__ellint_1 (_Tp __k, _Tp __phi)`

Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.

The incomplete elliptic integral of the first kind is defined as

$$F(k, \phi) = \int_0^\phi \frac{d\theta}{\sqrt{1 - k^2 \sin^2 \theta}}$$

Parameters

<code>__k</code>	The argument of the elliptic function.
<code>__phi</code>	The integral limit argument of the elliptic function.

Returns

The elliptic function of the first kind.

Definition at line 594 of file `sf_ellint.tcc`.

References `__comp_ellint_1()`, and `__ellint_rf()`.

8.3.2.84 `template<typename _Tp> _Tp std::__detail::__ellint_2 (_Tp __k, _Tp __phi)`

Return the incomplete elliptic integral of the second kind $E(k, \phi)$ using the Carlson formulation.

The incomplete elliptic integral of the second kind is defined as

$$E(k, \phi) = \int_0^\phi \sqrt{1 - k^2 \sin^2 \theta} d\theta$$

Parameters

<code>__k</code>	The argument of the elliptic function.
<code>__phi</code>	The integral limit argument of the elliptic function.

Returns

The elliptic function of the second kind.

Definition at line 673 of file `sf_ellint.tcc`.

References `__comp_ellint_2()`, `__ellint_rd()`, and `__ellint_rf()`.

8.3.2.85 `template<typename _Tp> _Tp std::__detail::__ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)`

Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ using the Carlson formulation.

The incomplete elliptic integral of the third kind is defined as

$$\Pi(k, \nu, \phi) = \int_0^\phi \frac{d\theta}{(1 - \nu \sin^2 \theta) \sqrt{1 - k^2 \sin^2 \theta}}$$

Parameters

<code>__k</code>	The argument of the elliptic function.
<code>__nu</code>	The second argument of the elliptic function.
<code>__phi</code>	The integral limit argument of the elliptic function.

Returns

The elliptic function of the third kind.

Definition at line 768 of file `sf_ellint.tcc`.

References `__comp_ellint_3()`, `__ellint_rf()`, and `__ellint_rj()`.

8.3.2.86 `template<typename _Tp> _Tp std::__detail::__ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`

Return the Bulirsch complete elliptic integrals.

Definition at line 920 of file sf_ellint.tcc.

References `__ellint_rf()`, and `__ellint_rj()`.

8.3.2.87 `template<typename _Tp> _Tp std::__detail::__ellint_d (_Tp __k, _Tp __phi)`

Return the Legendre elliptic integral D.

Definition at line 809 of file sf_ellint.tcc.

References `__ellint_rd()`.

8.3.2.88 `template<typename _Tp> _Tp std::__detail::__ellint_el1 (_Tp __x, _Tp __k_c)`

Return the Bulirsch elliptic integrals of the first kind.

Definition at line 848 of file sf_ellint.tcc.

References `__ellint_rf()`.

8.3.2.89 `template<typename _Tp> _Tp std::__detail::__ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`

Return the Bulirsch elliptic integrals of the second kind.

Definition at line 869 of file sf_ellint.tcc.

References `__ellint_rd()`, and `__ellint_rf()`.

8.3.2.90 `template<typename _Tp> _Tp std::__detail::__ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`

Return the Bulirsch elliptic integrals of the third kind.

Definition at line 894 of file sf_ellint.tcc.

References `__ellint_rf()`, and `__ellint_rj()`.

8.3.2.91 `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

<code>__x</code>	The first argument.
<code>__y</code>	The second argument.

Returns

The Carlson elliptic function.

Definition at line 81 of file `sf_ellint.tcc`.

Referenced by `__ellint_rf()`, and `__ellint_rj()`.

8.3.2.92 `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

<code>__x</code>	The first of two symmetric arguments.
<code>__y</code>	The second of two symmetric arguments.
<code>__z</code>	The third argument.

Returns

The Carlson elliptic function of the second kind.

Definition at line 163 of file `sf_ellint.tcc`.

Referenced by `__comp_ellint_2()`, `__comp_ellint_d()`, `__ellint_2()`, `__ellint_d()`, `__ellint_el2()`, `__ellint_rg()`, and `__ellint_rj()`.

8.3.2.93 `template<typename _Tp> _Tp std::__detail::__ellint_rf (_Tp __x, _Tp __y, _Tp __z)`

Return the Carlson elliptic function $R_F(x, y, z)$ of the first kind.

The Carlson elliptic function of the first kind is defined by:

$$R_F(x, y, z) = \frac{1}{2} \int_0^\infty \frac{dt}{(t+x)^{1/2}(t+y)^{1/2}(t+z)^{1/2}}$$

Parameters

<code>__x</code>	The first of three symmetric arguments.
<code>__y</code>	The second of three symmetric arguments.
<code>__z</code>	The third of three symmetric arguments.

Returns

The Carlson elliptic function of the first kind.

Definition at line 277 of file `sf_ellint.tcc`.

References `__comp_ellint_rf()`, and `__ellint_rc()`.

Referenced by `__comp_ellint_2()`, `__comp_ellint_3()`, `__ellint_1()`, `__ellint_2()`, `__ellint_3()`, `__ellint_cel()`, `__ellint_el1()`, `__ellint_el2()`, `__ellint_el3()`, and `__heuman_lambda()`.

8.3.2.94 `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 408 of file `sf_ellint.tcc`.

References `__comp_ellint_rg()`, and `__ellint_rd()`.

8.3.2.95 `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 456 of file `sf_ellint.tcc`.

References `__ellint_rc()`, and `__ellint_rd()`.

Referenced by `__comp_ellint_3()`, `__ellint_3()`, `__ellint_cel()`, `__ellint_el3()`, `__heuman_lambda()`, and `__jacobi_zeta()`.

8.3.2.96 `template<typename _Tp> _Tp std::__detail::__ellnome (_Tp __k)`

Return the elliptic nome given the modulus `k`.

Definition at line 292 of file `sf_theta.tcc`.

References `__ellnome_k()`, and `__ellnome_series()`.

Referenced by `__theta_c()`, `__theta_d()`, `__theta_n()`, and `__theta_s()`.

8.3.2.97 `template<typename _Tp> _Tp std::__detail::__ellnome_k (_Tp __k)`

Use the arithmetic-geometric mean to calculate the elliptic nome given the `k`.

Definition at line 278 of file `sf_theta.tcc`.

References `__comp_ellint_1()`.

Referenced by `__ellnome()`.

8.3.2.98 `template<typename _Tp> _Tp std::__detail::__ellnome_series (_Tp __k)`

Use MacLaurin series to calculate the elliptic nome given the `k`.

Definition at line 262 of file `sf_theta.tcc`.

Referenced by `__ellnome()`.

8.3.2.99 `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

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

This is something of an extension.

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 474 of file sf_expint.tcc.

References `__expint_E1()`, and `__expint_En_recursion()`.

Referenced by `__logint()`.

8.3.2.100 `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 514 of file sf_expint.tcc.

References `__expint_Ei()`.

8.3.2.101 `template<typename _Tp> _Tp std::__detail::__expint_asymp (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ for large argument.

The exponential integral is given by

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

This is something of an extension.

Parameters

<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 406 of file sf_expint.tcc.

8.3.2.102 `template<typename _Tp> _Tp std::__detail::__expint_E1 (_Tp __x)`

Return the exponential integral $E_1(x)$.

The exponential integral is given by

$$E_1(x) = \int_1^{\infty} \frac{e^{-xt}}{t} dt$$

Parameters

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Returns

The exponential integral.

Todo Find a good asymptotic switch point in $E_1(x)$.

Todo Find a good asymptotic switch point in $E_1(x)$.

Definition at line 375 of file `sf_expint.tcc`.

References `__expint_E1_asymp()`, `__expint_E1_series()`, `__expint_Ei()`, and `__expint_En_cont_frac()`.

Referenced by `__coshint()`, `__expint()`, `__expint_Ei()`, `__expint_En_recursion()`, and `__sinhint()`.

8.3.2.103 `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

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Returns

The exponential integral.

Definition at line 112 of file `sf_expint.tcc`.

Referenced by `__expint_E1()`.

8.3.2.104 `template<typename _Tp> _Tp std::__detail::__expint_E1_series (_Tp __x)`

Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.

The exponential integral is given by

$$E_1(x) = \int_1^{\infty} \frac{e^{-xt}}{t} dt$$

Parameters

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Returns

The exponential integral.

Definition at line 75 of file sf_expint.tcc.

Referenced by `__expint_E1()`.

8.3.2.105 `template<typename _Tp> _Tp std::__detail::__expint_Ei (_Tp __x)`

Return the exponential integral $Ei(x)$.

The exponential integral is given by

$$Ei(x) = - \int_{-x}^{\infty} \frac{e^t}{t} dt$$

Parameters

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Returns

The exponential integral.

Definition at line 351 of file sf_expint.tcc.

References `__expint_E1()`, `__expint_Ei_asyp()`, and `__expint_Ei_series()`.

Referenced by `__coshint()`, `__expint()`, `__expint_E1()`, and `__sinhint()`.

8.3.2.106 `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

<code>__x</code>	The argument of the exponential integral function.
------------------	--

Returns

The exponential integral.

Definition at line 318 of file sf_expint.tcc.

Referenced by `__expint_Ei()`.

8.3.2.107 `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 286 of file `sf_expint.tcc`.

Referenced by `__expint_Ei()`.

8.3.2.108 `template<typename _Tp> _Tp std::__detail::__expint_En_cont_frac (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ by continued fractions.

The exponential integral is given by

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n} dt$$

Parameters

<code>__n</code>	The order of the exponential integral function.
<code>__x</code>	The argument of the exponential integral function.

Returns

The exponential integral.

Definition at line 195 of file `sf_expint.tcc`.

Referenced by `__expint_E1()`.

8.3.2.109 `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

<code>__n</code>	The order of the exponential integral function.
<code>__x</code>	The argument of the exponential integral function.

Returns

The exponential integral.

Todo Find a principled starting number for the $E_n(x)$ downward recursion.

Definition at line 240 of file sf_expint.tcc.

References `__expint_E1()`.

Referenced by `__expint()`.

8.3.2.110 `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

<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 149 of file sf_expint.tcc.

References `__psi()`.

8.3.2.111 `template<typename _Tp> _Tp std::__detail::__expint_large_n (unsigned int __n, _Tp __x)`

Return the exponential integral $E_n(x)$ for large order.

The exponential integral is given by

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

This is something of an extension.

Parameters

<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 440 of file sf_expint.tcc.

8.3.2.112 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (unsigned int __n)`

Return the factorial of the integer n.

The factorial is:

$$n! = 12 \dots (n - 1)n, 0! = 1$$

Definition at line 2422 of file `sf_gamma.tcc`.

References `_S_factorial_table`.

Referenced by `__double_factorial()`.

8.3.2.113 `template<typename _Tp > _Tp std::__detail::__fermi_dirac (_Tp __s, _Tp __x)`

Return the Fermi-Dirac integral of real order s and real argument x.

See also

https://en.wikipedia.org/wiki/Clausen_function

<http://dlmf.nist.gov/25.12#iii>

Parameters

<code>__s</code>	The order $s \geq 0$.
<code>__x</code>	The real argument.

Returns

The real Fermi-Dirac cosine sum $F_s(x)$,

Definition at line 1379 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

8.3.2.114 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value 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 350 of file `sf_beta.tcc`.

References `__beta_inc()`.

8.3.2.115 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)`

Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value 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 379 of file `sf_beta.tcc`.

References `__beta_inc()`.

8.3.2.116 `template<typename _Tp > void std::__detail::__fock_airy (_Tp __x, std::complex<_Tp> & __w1, std::complex<_Tp> & __w2, std::complex<_Tp> & __w1p, std::complex<_Tp> & __w2p)`

Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w'_1(x)$ and $w'_2(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$

.

Parameters

<code>__x</code>	The argument of the Airy functions.
<code>__w1</code>	The output Fock-type Airy function of the first kind.
<code>__w2</code>	The output Fock-type Airy function of the second kind.
<code>__w1p</code>	The output derivative of the Fock-type Airy function of the first kind.
<code>__w2p</code>	The output derivative of the Fock-type Airy function of the second kind.

Definition at line 580 of file `sf_mod_bessel.tcc`.

8.3.2.117 `template<typename _Tp > bool std::__detail::__fpequal (const _Tp & __a, const _Tp & __b)`

A function to reliably compare two floating point numbers.

Parameters

<code>__a</code>	the left hand side.
<code>__b</code>	the right hand side

Returns

returns true if a and b are equal to zero or differ only by $\max(a, b) * 5 * \epsilon$

Definition at line 62 of file `sf_polylog.tcc`.

Referenced by `__dirichlet_beta()`, `__dirichlet_eta()`, `__fpimag()`, `__fpreal()`, `__polylog()`, `__polylog_exp_asymp()`, `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_neg()`, `__polylog_exp_neg_even()`, `__polylog_exp_neg_odd()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__polylog_exp_real_pos()`.

8.3.2.118 `template<typename _Tp> bool std::__detail::__fpimag (const std::complex< _Tp> & __w)`

A function to reliably test a complex number for imaginyness [?].

Parameters

<code>__w</code>	The complex argument.
------------------	-----------------------

Returns

`true` if $Re(w)$ is zero within $5 * epsilon$, `false` otherwise.

Definition at line 107 of file `sf_polylog.tcc`.

References `__fpequal()`.

8.3.2.119 `template<typename _Tp> bool std::__detail::__fpimag (const _Tp)`

Definition at line 117 of file `sf_polylog.tcc`.

8.3.2.120 `template<typename _Tp> bool std::__detail::__fpreal (const std::complex< _Tp> & __w)`

A function to reliably test a complex number for realness.

Parameters

<code>__w</code>	The complex argument.
------------------	-----------------------

Returns

`true` if $Im(w)$ is zero within $5 * epsilon$, `false` otherwise.

Definition at line 84 of file `sf_polylog.tcc`.

References `__fpequal()`.

Referenced by `__polylog_exp_int_pos()`, and `__polylog_exp_real_pos()`.

8.3.2.121 `template<typename _Tp> bool std::__detail::__fpreal (const _Tp)`

Definition at line 94 of file `sf_polylog.tcc`.

8.3.2.122 `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

__x	The argument
-----	--------------

Definition at line 166 of file sf_fresnel.tcc.

References `__fresnel_cont_frac()`, and `__fresnel_series()`.

8.3.2.123 `template<typename _Tp> void std::__detail::__fresnel_cont_frac (const _Tp __ax, _Tp & _Cf, _Tp & _Sf)`

This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.

Definition at line 105 of file sf_fresnel.tcc.

Referenced by `__fresnel()`.

8.3.2.124 `template<typename _Tp> void std::__detail::__fresnel_series (const _Tp __ax, _Tp & _Cf, _Tp & _Sf)`

This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.

Definition at line 48 of file sf_fresnel.tcc.

Referenced by `__fresnel()`.

8.3.2.125 `template<typename _Tp> _Tp std::__detail::__gamma (_Tp __x)`

Return $\Gamma(x)$.

Parameters

__x	The argument of the gamma function.
-----	-------------------------------------

Returns

The gamma function.

Definition at line 1918 of file sf_gamma.tcc.

References `__log_gamma()`.

Referenced by `__beta_gamma()`, and `__riemann_zeta()`.

8.3.2.126 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma_cont_frac (_Tp __a, _Tp __x)`

Definition at line 1964 of file sf_gamma.tcc.

Referenced by `__gamma_l()`, `__gamma_p()`, `__gamma_q()`, and `__gamma_u()`.

8.3.2.127 `template<typename _Tp> _Tp std::__detail::__gamma_l (_Tp __a, _Tp __x)`

Return the lower incomplete gamma function. The lower incomplete gamma function is defined by

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

.

Definition at line 2070 of file sf_gamma.tcc.

References `__gamma_cont_frac()`, and `__gamma_series()`.

8.3.2.128 `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 2013 of file sf_gamma.tcc.

References `__gamma_cont_frac()`, and `__gamma_series()`.

Referenced by `__chi_squared_pdf()`.

8.3.2.129 `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 2044 of file sf_gamma.tcc.

References `__gamma_cont_frac()`, and `__gamma_series()`.

Referenced by `__chi_squared_pdfc()`.

8.3.2.130 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__gamma_series (_Tp __a, _Tp __x)`

Definition at line 1930 of file sf_gamma.tcc.

Referenced by `__gamma_l()`, `__gamma_p()`, `__gamma_q()`, and `__gamma_u()`.

8.3.2.131 `template<typename _Tp> void std::__detail::__gamma_temme (_Tp __mu, _Tp & __gam1, _Tp & __gam2, _Tp & __gampl, _Tp & __gammi)`

Compute the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ is $\mu = \nu - N$ and N is the nearest integer to ν . The values of $\Gamma(1+\mu)$ and $\Gamma(1-\mu)$ are returned as well.

The accuracy requirements on this are exquisite.

Parameters

	<code>__mu</code>	The input parameter of the gamma functions.
out	<code>__gam1</code>	The output function $\Gamma_1(\mu)$
out	<code>__gam2</code>	The output function $\Gamma_2(\mu)$
out	<code>__gampl</code>	The output function $\Gamma(1+\mu)$
out	<code>__gammi</code>	The output function $\Gamma(1-\mu)$

Definition at line 163 of file `sf_bessel.tcc`.

Referenced by `__cyl_bessel_ik_steel()`, and `__cyl_bessel_jn_steel()`.

8.3.2.132 `template<typename _Tp> _Tp std::__detail::__gamma_u (_Tp __a, _Tp __x)`

Return the upper incomplete gamma function. The lower incomplete gamma function is defined by

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1} dt (a > 0)$$

.

Definition at line 2102 of file `sf_gamma.tcc`.

References `__gamma_cont_frac()`, and `__gamma_series()`.

8.3.2.133 `template<typename _Tp> _Tp std::__detail::__gauss (_Tp __x)`

The CDF of the normal distribution. i.e. the integrated lower tail of the normal PDF.

Definition at line 70 of file `sf_owens_t.tcc`.

8.3.2.134 `template<typename _Tp> _Tp std::__detail::__gegenbauer_poly (unsigned int __n, _Tp __alpha, _Tp __x)`

Return the Gegenbauer polynomial $C_n^\alpha(x)$ of degree n and real order α and argument x .

The Gegenbauer polynomials are generated by a three-term recursion relation:

$$C_n^\alpha(x) = \frac{1}{n} [2x(n+\alpha-1)C_{n-1}^\alpha(x) - (n+2\alpha-2)C_{n-2}^\alpha(x)]$$

and $C_0^\alpha(x) = 1$, $C_1^\alpha(x) = 2\alpha x$.

Template Parameters

<code>__Talpha</code>	The real type of the order
<code>__Tp</code>	The real type of the argument

Parameters

<code>__n</code>	The non-negative integral degree
<code>__alpha</code>	The real order
<code>__x</code>	The real argument

Definition at line 61 of file `sf_gegenbauer.tcc`.

```
8.3.2.135 template<typename _Tp> void std::__detail::__hankel ( std::complex<_Tp> __nu, std::complex<_Tp> __z,
std::complex<_Tp> &_H1, std::complex<_Tp> &_H2, std::complex<_Tp> &_H1p, std::complex<_Tp> &_H2p
)
```

Parameters

in	<code>__nu</code>	The order for which the Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the Hankel functions are evaluated.
out	<code>_H1</code>	The Hankel function of the first kind.
out	<code>_H1p</code>	The derivative of the Hankel function of the first kind.
out	<code>_H2</code>	The Hankel function of the second kind.
out	<code>_H2p</code>	The derivative of the Hankel function of the second kind.

Definition at line 1127 of file `sf_hankel.tcc`.

References `__debye_region()`, `__hankel_debye()`, and `__hankel_uniform()`.

Referenced by `__cyl_bessel()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann()`, and `__sph_hankel()`.

```
8.3.2.136 template<typename _Tp> void std::__detail::__hankel_debye ( std::complex<_Tp> __nu, std::complex<_Tp> __z,
std::complex<_Tp> __alpha, int __indexr, char &__aorb, int &__morn, std::complex<_Tp> &_H1, std::complex<
_Tp> &_H2, std::complex<_Tp> &_H1p, std::complex<_Tp> &_H2p )
```

Parameters

in	<code>__nu</code>	The order for which the Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the Hankel functions are evaluated.
in	<code>__alpha</code>	
in	<code>__indexr</code>	
out	<code>__aorb</code>	
out	<code>__morn</code>	
out	<code>_H1</code>	The Hankel function of the first kind.
out	<code>_H1p</code>	The derivative of the Hankel function of the first kind.
out	<code>_H2</code>	The Hankel function of the second kind.
out	<code>_H2p</code>	The derivative of the Hankel function of the second kind.

Definition at line 959 of file `sf_hankel.tcc`.

Referenced by `__hankel()`.

8.3.2.137 `template<typename _Tp> void std::__detail::__hankel_params (std::complex<_Tp> __nu, std::complex<_Tp> __zhat, std::complex<_Tp> & __p, std::complex<_Tp> & __p2, std::complex<_Tp> & __nup2, std::complex<_Tp> & __num2, std::complex<_Tp> & __num1d3, std::complex<_Tp> & __num2d3, std::complex<_Tp> & __num4d3, std::complex<_Tp> & __zeta, std::complex<_Tp> & __zetaphf, std::complex<_Tp> & __zetamhf, std::complex<_Tp> & __zetam3hf, std::complex<_Tp> & __zetrat)`

Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.

Definition at line 110 of file `sf_hankel.tcc`.

Referenced by `__hankel_uniform_outer()`.

8.3.2.138 `template<typename _Tp> void std::__detail::__hankel_uniform (std::complex<_Tp> __nu, std::complex<_Tp> __z, std::complex<_Tp> & __H1, std::complex<_Tp> & __H2, std::complex<_Tp> & __H1p, std::complex<_Tp> & __H2p)`

This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.

Parameters

in	<code>__nu</code>	The order for which the Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the Hankel functions are evaluated.
out	<code>__H1</code>	The Hankel function of the first kind.
out	<code>__H1p</code>	The derivative of the Hankel function of the first kind.
out	<code>__H2</code>	The Hankel function of the second kind.
out	<code>__H2p</code>	The derivative of the Hankel function of the second kind.

Definition at line 904 of file `sf_hankel.tcc`.

References `__hankel_uniform_olver()`.

Referenced by `__hankel()`.

8.3.2.139 `template<typename _Tp> void std::__detail::__hankel_uniform_olver (std::complex<_Tp> __nu, std::complex<_Tp> __z, std::complex<_Tp> & __H1, std::complex<_Tp> & __H2, std::complex<_Tp> & __H1p, std::complex<_Tp> & __H2p)`

Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order `nu` along with their derivatives.

Parameters

in	<code>__nu</code>	The order for which the Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the Hankel functions are evaluated.
out	<code>__H1</code>	The Hankel function of the first kind.
out	<code>__H1p</code>	The derivative of the Hankel function of the first kind.
out	<code>__H2</code>	The Hankel function of the second kind.
out	<code>__H2p</code>	The derivative of the Hankel function of the second kind.

Definition at line 818 of file `sf_hankel.tcc`.

References `__hankel_uniform_outer()`, and `__hankel_uniform_sum()`.

Referenced by `__hankel_uniform()`.

```
8.3.2.140 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 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 273 of file `sf_hankel.tcc`.

References `__airy()`, `__airy_arg()`, and `__hankel_params()`.

Referenced by `__hankel_uniform_olver()`.

```
8.3.2.141 template<typename _Tp> void std::__detail::__hankel_uniform_sum ( std::complex<_Tp> __p, std::complex<_Tp> __p2, std::complex<_Tp> __num2, std::complex<_Tp> __zetam3hf, std::complex<_Tp> __Aip, std::complex<_Tp> __o4dp, std::complex<_Tp> __Aim, std::complex<_Tp> __o4dm, std::complex<_Tp> __od2p, std::complex<_Tp> __od0dp, std::complex<_Tp> __od2m, std::complex<_Tp> __od0dm, _Tp __eps, std::complex<_Tp> & __H1sum, std::complex<_Tp> & __H1psum, std::complex<_Tp> & __H2sum, std::complex<_Tp> & __H2psum )
```

Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to `nterms` (less than 5) to achieve relative error `eps`.

Parameters

in	<code>__p</code>	
in	<code>__p2</code>	
in	<code>__num2</code>	
in	<code>__zetam3hf</code>	
in	<code>__Aip</code>	The Airy function value $Ai()$.
in	<code>__o4dp</code>	
in	<code>__Aim</code>	The Airy function value $Ai()$.
in	<code>__o4dm</code>	
in	<code>__od2p</code>	
in	<code>__od0dp</code>	
in	<code>__od2m</code>	
in	<code>__od0dm</code>	
in	<code>__eps</code>	The error tolerance
out	<code>__H1sum</code>	The Hankel function of the first kind.
out	<code>__H1psum</code>	The derivative of the Hankel function of the first kind.
out	<code>__H2sum</code>	The Hankel function of the second kind.
out	<code>__H2psum</code>	The derivative of the Hankel function of the second kind.

Definition at line 351 of file `sf_hankel.tcc`.

Referenced by `__hankel_uniform_olver()`.

```
8.3.2.142 template<typename _Tp> _Tp std::__detail::__heuman_lambda ( _Tp __k, _Tp __phi )
```

Return the Heuman lambda function.

Definition at line 941 of file sf_ellint.tcc.

References `__ellint_rf()`, and `__ellint_rj()`.

8.3.2.143 `template<typename _Tp> _Tp std::__detail::__hurwitz_zeta (_Tp __s, _Tp __a)`

Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.

The Hurwitz zeta function is defined by:

$$\zeta(s, a) = \sum_{n=0}^{\infty} \frac{1}{(n+a)^s}$$

The Riemann zeta function is a special case:

$$\zeta(s) = \zeta(s, 1)$$

Parameters

<code>__s</code>	The argument $s \neq 1$
<code>__a</code>	The scale parameter $a > -1$

Definition at line 702 of file sf_zeta.tcc.

References `__hurwitz_zeta_euler_maclaurin()`.

8.3.2.144 `template<typename _Tp> std::complex<_Tp> std::__detail::__hurwitz_zeta (_Tp __s, std::complex<_Tp> __a)`

Return the Hurwitz Zeta function for real s and complex a .

Parameters

<code>__s</code>	The real argument
<code>__a</code>	The complex parameter

Todo This `__hurwitz_zeta` prefactor is prone to overflow. positive integer orders s ?

Definition at line 1119 of file sf_polylog.tcc.

References `__polylog_exp()`.

Referenced by `__psi()`.

8.3.2.145 `template<typename _Tp> _Tp std::__detail::__hurwitz_zeta_euler_maclaurin (_Tp __s, _Tp __a)`

Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.

See also

An efficient algorithm for accelerating the convergence of oscillatory series, useful for computing the polylogarithm and Hurwitz zeta functions, Linas Vep

Parameters

<code>__s</code>	The argument $s! = 1$
<code>__a</code>	The scale parameter $a > -1$

Definition at line 560 of file `sf_zeta.tcc`.

References `_S_Euler_Maclaurin_zeta`.

Referenced by `__hurwitz_zeta()`.

8.3.2.146 `template<typename _Tp> std::complex<_Tp> std::__detail::__hydrogen (const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp __Z, const _Tp __r, const _Tp __theta, const _Tp __phi)`

Definition at line 44 of file `sf_hydrogen.tcc`.

References `__assoc_laguerre()`, `__psi()`, and `__sph_legendre()`.

8.3.2.147 `template<typename _Tp> _Tp std::__detail::__hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$.

The hypergeometric function is defined by

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

Parameters

<code>__a</code>	The first <i>numerator</i> parameter.
<code>__b</code>	The second <i>numerator</i> parameter.
<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 776 of file `sf_hyperg.tcc`.

References `__hyperg_luke()`, `__hyperg_reflect()`, `__hyperg_series()`, `__log_gamma()`, and `__log_gamma_sign()`.

8.3.2.148 `template<typename _Tp> _Tp std::__detail::__hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.

Definition at line 352 of file `sf_hyperg.tcc`.

Referenced by `__hyperg()`.

8.3.2.149 `template<typename _Tp> _Tp std::__detail::__hyperg_reflect (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by the reflection formulae in Abramowitz & Stegun formula 15.3.6 for $d = c - a - b$ not integral and formula 15.3.11 for $d = c - a - b$ integral. This assumes $a, b, c \neq$ negative integer.

The hypergeometric function is defined by

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

The reflection formula for nonintegral $d = c - a - b$ is:

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)\Gamma(d)}{\Gamma(c-a)\Gamma(c-b)} {}_2F_1(a, b; 1-d; 1-x) + \frac{\Gamma(c)\Gamma(-d)}{\Gamma(a)\Gamma(b)} {}_2F_1(c-a, c-b; 1+d; 1-x)$$

The reflection formula for integral $m = c - a - b$ is:

$${}_2F_1(a, b; a+b+m; x) = \frac{\Gamma(m)\Gamma(a+b+m)}{\Gamma(a+m)\Gamma(b+m)} \sum_{k=0}^{m-1} \frac{(m+a)_k(m+b)_k}{k!(1-m)_k} -$$

Definition at line 486 of file sf_hyperg.tcc.

References __hyperg_series(), __log_gamma(), __log_gamma_sign(), and __psi().

Referenced by __hyperg().

8.3.2.150 `template<typename _Tp> _Tp std::__detail::__hyperg_series (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by series expansion.

The hypergeometric function is defined by

$${}_2F_1(a, b; c; x) = \frac{\Gamma(c)}{\Gamma(a)\Gamma(b)} \sum_{n=0}^{\infty} \frac{\Gamma(a+n)\Gamma(b+n)}{\Gamma(c+n)} \frac{x^n}{n!}$$

This works and it's pretty fast.

Parameters

<code>__a</code>	The first <i>numerator</i> parameter.
<code>__b</code>	The second <i>numerator</i> parameter.
<code>__c</code>	The <i>denominator</i> parameter.
<code>__x</code>	The argument of the confluent hypergeometric function.

Returns

The confluent hypergeometric function.

Definition at line 321 of file sf_hyperg.tcc.

Referenced by __hyperg(), and __hyperg_reflect().

8.3.2.151 `template<typename _Tp> std::tuple<_Tp, _Tp, _Tp> std::__detail::__jacobi_sncndn (_Tp __k, _Tp __u)`

Return a tuple of the three primary Jacobi elliptic functions: $sn(k, u)$, $cn(k, u)$, $dn(k, u)$.

Definition at line 414 of file sf_theta.tcc.

8.3.2.152 `template<typename _Tp> _Tp std::__detail::__jacobi_zeta (_Tp __k, _Tp __phi)`

Return the Jacobi zeta function.

Definition at line 971 of file `sf_ellint.tcc`.

References `__comp_ellint_1()`, and `__ellint_rj()`.

8.3.2.153 `template<typename _Tp> _Tp std::__detail::__laguerre (unsigned int __n, _Tp __x)`

This routine returns the Laguerre polynomial of order n : $L_n(x)$.

The Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})$$

Parameters

<code>__n</code>	The order of the Laguerre polynomial.
<code>__x</code>	The argument of the Laguerre polynomial.

Returns

The value of the Laguerre polynomial of order n and argument x .

Definition at line 321 of file `sf_laguerre.tcc`.

8.3.2.154 `template<typename _Tp> _Tp std::__detail::__legendre_q (unsigned int __l, _Tp __x)`

Return the Legendre function of the second kind by upward recursion on order l .

The Legendre function of the second kind of order l and argument x , $Q_l(x)$, is defined by:

$$Q_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} (x^2 - 1)^l$$

Parameters

<code>__l</code>	The order of the Legendre function. $l \geq 0$.
<code>__x</code>	The argument of the Legendre function. $ x \leq 1$.

Definition at line 123 of file `sf_legendre.tcc`.

8.3.2.155 `template<typename _Tp> _Tp std::__detail::__log_bincoef (unsigned int __n, unsigned int __k)`

Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

.

Parameters

<code>__n</code>	The first argument of the binomial coefficient.
<code>__k</code>	The second argument of the binomial coefficient.

Returns

The logarithm of the binomial coefficient.

Definition at line 1862 of file sf_gamma.tcc.

8.3.2.156 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (_Tp __x)`

Definition at line 2450 of file sf_gamma.tcc.

References `__log_gamma()`.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

8.3.2.157 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (int __n)`

Return the logarithm of the double factorial of the integer n.

The double factorial is defined for integral n by:

$$n!! = 135\dots(n-2)n, \text{ odd } n!! = 246\dots(n-2)n, \text{ even } n!! = 10!! = 1$$

The double factorial is defined for odd negative integers in the obvious way:

$$(-2m-1)!! = 1/(1(-1)(-3)\dots(-2m+1)(-2m-1)) = \frac{(-1)^m}{(2m-1)!!}$$

for $n = -2m - 1$.

Definition at line 2516 of file sf_gamma.tcc.

References `__log_double_factorial()`, `__log_factorial()`, `_S_double_factorial_table`, and `_S_neg_double_factorial_table`.

8.3.2.158 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_factorial (unsigned int __n)`

Return the logarithm of the factorial of the integer n.

The factorial is:

$$n! = 12\dots(n-1)n, 0! = 1$$

Definition at line 2440 of file sf_gamma.tcc.

References `__log_gamma()`, and `_S_factorial_table`.

Referenced by `__log_double_factorial()`.

8.3.2.159 `template<typename _Tp > _Tp std::__detail::__log_gamma (_Tp __x)`

Return $\log(|\Gamma(x)|)$. This will return values even for $x < 0$. To recover the sign of $\Gamma(x)$ for any argument use `__log_gamma_sign`.

Parameters

<code>__x</code>	The argument of the log of the gamma function.
------------------	--

Returns

The logarithm of the gamma function.

Definition at line 1800 of file `sf_gamma.tcc`.

References `__log_gamma_lanczos()`.

Referenced by `__beta_lgamma()`, `__cyl_bessel_ij_series()`, `__gamma()`, `__hyperg()`, `__hyperg_reflect()`, `__log_double_factorial()`, `__log_factorial()`, `__log_pochhammer_u()`, `__poly_laguerre_large_n()`, `__psi()`, `__riemann_zeta()`, `__riemann_zeta_glob()`, and `__sph_legendre()`.

8.3.2.160 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (_Tp __x)`

Return $\log(\Gamma(x))$ by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.

Parameters

<code>__x</code>	The argument of the log of the gamma function.
------------------	--

Returns

The logarithm of the gamma function.

Definition at line 1699 of file `sf_gamma.tcc`.

8.3.2.161 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_lanczos (_Tp __x)`

Return $\log(\Gamma(x))$ by the Lanczos method. This method dominates all others on the positive axis I think.

Parameters

<code>__x</code>	The argument of the log of the gamma function.
------------------	--

Returns

The logarithm of the gamma function.

Definition at line 1755 of file `sf_gamma.tcc`.

Referenced by `__log_gamma()`.

8.3.2.162 `template<typename _Tp > _Tp std::__detail::__log_gamma_sign (_Tp __x)`

Return the sign of $\Gamma(x)$. At nonpositive integers zero is returned.

Parameters

<code>__x</code>	The argument of the gamma function.
------------------	-------------------------------------

Returns

The sign of the gamma function.

Definition at line 1831 of file sf_gamma.tcc.

Referenced by `__hyperg()`, `__hyperg_reflect()`, and `__pochhammer_l()`.

8.3.2.163 `template<typename _Tp> _GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_spouge (_Tp __z)`

Return $\Gamma(z)$ by the Spouge algorithm:

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[\sqrt{2\pi} \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

See also

Spouge, J.L., Computation of the gamma, digamma, and trigamma functions. SIAM Journal on Numerical Analysis 31, 3 (1994), pp. 931-944

Parameters

<code>__z</code>	The argument of the gamma function.
------------------	-------------------------------------

Returns

The the gamma function.

Definition at line 1739 of file sf_gamma.tcc.

8.3.2.164 `template<typename _Tp> _Tp std::__detail::__log_pochhammer_l (_Tp __a, _Tp __n)`

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular, $(n)_n = n!$. Thus this function returns

$$\ln[(a)_n] = \Gamma(a+1) - \Gamma(a-n+1), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left\{ \begin{matrix} a \\ n \end{matrix} \right\}$$

, and others.

Definition at line 2209 of file sf_gamma.tcc.

8.3.2.165 `template<typename _Tp> _Tp std::__detail::__log_pochhammer_u(_Tp __a, _Tp __n)`

Return the logarithm of the (upper) Pochhammer symbol or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(n)$$

Thus this function returns

$$\ln[(a)_n] = \Gamma(a+n) - \Gamma(n), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

Definition at line 2144 of file sf_gamma.tcc.

References `__log_gamma()`.

8.3.2.166 `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 535 of file sf_expint.tcc.

References `__expint()`.

8.3.2.167 `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	<code>__h</code>	The scale parameter.
in	<code>__a</code>	The integration limit.

Returns

The owens T function.

Definition at line 92 of file `sf_owens_t.tcc`.

References `__znorm1()`, and `__znorm2()`.

8.3.2.168 `template<typename _Tp> _Tp std::__detail::__pochhammer_l (_Tp __a, _Tp __n)`

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a - k), (a)_0 = 1 = \Gamma(a + 1) / \Gamma(a - n + 1)$$

In particular, $(n)_n = n!$.

Definition at line 2232 of file `sf_gamma.tcc`.

References `__log_gamma_sign()`.

8.3.2.169 `template<typename _Tp> _Tp std::__detail::__pochhammer_u (_Tp __a, _Tp __n)`

Return the (upper) Pochhammer function or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a + k), (a)_0 = 1 = \Gamma(a + n) / \Gamma(a)$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

Definition at line 2170 of file `sf_gamma.tcc`.

8.3.2.170 `template<typename _Tp> _Tp std::__detail::__poly_hermite (unsigned int __n, _Tp __x)`

This routine returns the Hermite polynomial of order n: $H_n(x)$.

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

The Hermite polynomial obeys a reflection formula:

$$H_n(-x) = (-1)^n H_n(x)$$

Parameters

<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 179 of file `sf_hermite.tcc`.

References `__poly_hermite_asymp()`, and `__poly_hermite_recursion()`.

8.3.2.171 `template<typename _Tp> _Tp std::__detail::__poly_hermite_asymp (unsigned int __n, _Tp __x)`

This routine returns the Hermite polynomial of large order n : $H_n(x)$. We assume here that $x \geq 0$.

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

see "Asymptotic analysis of the Hermite polynomials from their differential-difference equation", Diego Dominici, arXiv↵
:math/0601078v1 [math.CA] 4 Jan 2006

Parameters

<code>__n</code>	The order of the Hermite polynomial.
<code>__x</code>	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x .

Definition at line 113 of file `sf_hermite.tcc`.

References `__airy()`.

Referenced by `__poly_hermite()`.

8.3.2.172 `template<typename _Tp> _Tp std::__detail::__poly_hermite_recursion (unsigned int __n, _Tp __x)`

This routine returns the Hermite polynomial of order n : $H_n(x)$ by recursion on n .

The Hermite polynomial is defined by:

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}$$

Parameters

<code>__n</code>	The order of the Hermite polynomial.
<code>__x</code>	The argument of the Hermite polynomial.

Returns

The value of the Hermite polynomial of order n and argument x .

Definition at line 69 of file `sf_hermite.tcc`.

Referenced by `__poly_hermite()`.

8.3.2.173 `template<typename _Tp> _Tp std::__detail::__poly_jacobi (unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)`

Compute the Jacobi polynomial by recursion on n :

$$2n(\alpha+\beta+n)(\alpha+\beta+2n-2)P_n^{(\alpha,\beta)}(x) = (\alpha+\beta+2n-1)((\alpha^2-\beta^2)+x(\alpha+\beta+2n-2)(\alpha+\beta+2n))P_{n-1}^{(\alpha,\beta)}(x) - 2(\alpha+n-1)(\beta+n-1)(\alpha+\beta+2n-2)P_{n-2}^{(\alpha,\beta)}(x)$$

Definition at line 57 of file sf_jacobi.tcc.

References `__beta()`.

Referenced by `__poly_radial_jacobi()`.

8.3.2.174 `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre (unsigned int __n, _Tpa __alpha1, _Tp __x)`

This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^\alpha lpha(x)$.

The associated Laguerre function is defined by

$$L_n^\alpha(x) = \frac{(\alpha+1)_n}{n!} {}_1F_1(-n; \alpha+1; x)$$

where $(\alpha)_n$ is the Pochhammer symbol and ${}_1F_1(a; c; x)$ is the confluent hypergeometric function.

The associated Laguerre polynomial is defined for integral $\alpha = m$ by:

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x)$$

where the Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})$$

Template Parameters

<code>_Tpa</code>	The type of the degree.
<code>_Tp</code>	The type of the parameter.

Parameters

<code>__n</code>	The order of the Laguerre function.
<code>__alpha1</code>	The degree of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

Returns

The value of the Laguerre function of order n , degree α , and argument x .

Definition at line 248 of file sf_laguerre.tcc.

References `__poly_laguerre_hyperg()`, `__poly_laguerre_large_n()`, and `__poly_laguerre_recursion()`.

8.3.2.175 `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre_hyperg (unsigned int __n, _Tpa __alpha1, _Tp __x)`

Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.

The associated Laguerre function is defined by

$$L_n^\alpha(x) = \frac{(\alpha+1)_n}{n!} {}_1F_1(-n; \alpha+1; x)$$

where $(\alpha)_n$ is the Pochhammer symbol and ${}_1F_1(a; c; x)$ is the confluent hypergeometric function.

This function assumes $x \neq 0$.

This is from the GNU Scientific Library.

Template Parameters

<code>_Tpa</code>	The type of the degree.
<code>_Tp</code>	The type of the parameter.

Parameters

<code>__n</code>	The order of the Laguerre function.
<code>__alpha1</code>	The degree of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

Returns

The value of the Laguerre function of order n , degree α , and argument x .

Definition at line 129 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

8.3.2.176 `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre_large_n (unsigned __n, _Tpa __alpha1, _Tp __x)`

This routine returns the associated Laguerre polynomial of order n , degree $\alpha > -1$ for large n . Abramowitz & Stegun, 13.5.21.

Template Parameters

<code>_Tpa</code>	The type of the degree.
<code>_Tp</code>	The type of the parameter.

Parameters

<code>__n</code>	The order of the Laguerre function.
<code>__alpha1</code>	The degree of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

Returns

The value of the Laguerre function of order n , degree α , and argument x .

This is from the GNU Scientific Library.

Definition at line 72 of file `sf_laguerre.tcc`.

References `__log_gamma()`.

Referenced by `__poly_laguerre()`.

8.3.2.177 `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre_recursion (unsigned int __n, _Tpa __alpha1, _Tp __x)`

This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^\alpha(x)$ by recursion.

The associated Laguerre function is defined by

$$L_n^\alpha(x) = \frac{(\alpha+1)_n}{n!} {}_1F_1(-n; \alpha+1; x)$$

where $(\alpha)_n$ is the Pochhammer symbol and ${}_1F_1(a; c; x)$ is the confluent hypergeometric function.

The associated Laguerre polynomial is defined for integral $\alpha = m$ by:

$$L_n^m(x) = (-1)^m \frac{d^m}{dx^m} L_{n+m}(x)$$

where the Laguerre polynomial is defined by:

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})$$

Template Parameters

<code>_Tpa</code>	The type of the degree.
<code>_Tp</code>	The type of the parameter.

Parameters

<code>__n</code>	The order of the Laguerre function.
<code>__alpha1</code>	The degree of the Laguerre function.
<code>__x</code>	The argument of the Laguerre function.

Returns

The value of the Laguerre function of order n , degree α , and argument x .

Definition at line 187 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

8.3.2.178 `template<typename _Tp> _Tp std::__detail::__poly_legendre_p (unsigned int __l, _Tp __x)`

Return the Legendre polynomial by upward recursion on order l .

The Legendre function of order l and argument x , $P_l(x)$, is defined by:

$$P_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} (x^2 - 1)^l$$

Parameters

<code>__l</code>	The order of the Legendre polynomial. $l \geq 0$.
------------------	--

<code>__x</code>	The argument of the Legendre polynomial. $ x \leq 1$.
------------------	---

Definition at line 73 of file `sf_legendre.tcc`.

Referenced by `__assoc_legendre_p()`, and `__sph_legendre()`.

8.3.2.179 `template<typename _Tp> _Tp std::__detail::__poly_radial_jacobi (unsigned int __n, unsigned int __m, _Tp __rho)`

Return the radial polynomial $R_n^m(\rho)$ for non-negative degree n , order $m \leq n$, and real radial argument ρ .

The radial polynomials are defined by

$$R_n^m(\rho) = \sum_{k=0}^{\frac{n-m}{2}} \frac{(-1)^k (n-k)!}{k! \left(\frac{n+m}{2} - k\right)! \left(\frac{n-m}{2} - k\right)!} \rho^{n-2k}$$

for $n - m$ even and identically 0 for $n - m$ odd. The radial polynomials can be related to the Zernike polynomials:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative m, n .

See also

`zernike` for details on the Zernike polynomials.

Template Parameters

<code>_Tp</code>	The real type of the radial coordinate
------------------	--

Parameters

<code>__n</code>	The non-negative degree.
<code>__m</code>	The non-negative azimuthal order
<code>__rho</code>	The radial argument

Definition at line 138 of file `sf_jacobi.tcc`.

References `__poly_jacobi()`.

Referenced by `__zernike()`, `__gnu_cxx::radpolyf()`, and `__gnu_cxx::radpolyf()`.

8.3.2.180 `template<typename _Tp> _Tp std::__detail::__polylog (_Tp __s, _Tp __x)`

Return the polylog $\text{Li}_s(x)$ for two real arguments.

Parameters

<code>__s</code>	The real index.
<code>__x</code>	The real argument.

Returns

The complex value of the polylogarithm.

Definition at line 1072 of file `sf_polylog.tcc`.

References `__fpequal()`, and `__polylog_exp()`.

Referenced by `__dirichlet_beta()`, `__dirichlet_eta()`, and `__polylog()`.

8.3.2.181 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog (_Tp __s, std::complex<_Tp> __w)`

Return the polylog in those cases where we can calculate it.

Parameters

<code>__s</code>	The real index.
<code>__w</code>	The complex argument.

Returns

The complex value of the polylogarithm.

Definition at line 1102 of file sf_polylog.tcc.

References `__fpequal()`, `__polylog()`, and `__polylog_exp()`.

8.3.2.182 `template<typename _Tp, typename ArgType> __gnu_cxx::__promote_num_t<std::complex<_Tp>, ArgType> std::__detail::__polylog_exp (_Tp __s, ArgType __w)`

This is the frontend function which calculates $Li_s(e^w)$. First we branch into different parts depending on the properties of s . This function is the same irrespective of a real or complex w , hence the template parameter `ArgType`.

Note

: I *really* wish we could return a variant<Tp, std::complex<Tp>>.

Parameters

<code>__s</code>	The real order.
<code>__w</code>	The real or complex argument.

Returns

The real or complex value of $Li_s(e^w)$.

Definition at line 1039 of file sf_polylog.tcc.

References `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

Referenced by `__bose_einstein()`, `__clausen()`, `__clausen_c()`, `__clausen_s()`, `__fermi_dirac()`, `__hurwitz_zeta()`, and `__polylog()`.

8.3.2.183 `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 u the imaginary part of the polylog is given by $Im(Li_s(e^u)) = -\pi u^{s-1} / \Gamma(s)$. Check this relation for any benchmark that you use. The use of `evenzeta` leads to a speedup of about 1000.

Parameters

<code>__s</code>	the real index s .
<code>__w</code>	the large complex argument w .

Returns

the value of the polylogarithm.

Definition at line 686 of file `sf_polylog.tcc`.

References `__fpequal()`.

Referenced by `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

```
8.3.2.184  template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_neg ( int __s, std::complex<_Tp>
           __w )
```

This treats the case where s is a negative integer.

Parameters

<code>__s</code>	a negative integer.
<code>__w</code>	an arbitrary complex number

Returns

the value of the polylogarithm.

Definition at line 856 of file `sf_polylog.tcc`.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

Referenced by `__polylog_exp()`.

```
8.3.2.185  template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_neg ( const int __s, _Tp __w )
```

This treats the case where s is a negative integer and w is a real.

Parameters

<code>__s</code>	a negative integer.
<code>__w</code>	the argument.

Returns

the value of the polylogarithm.

Definition at line 898 of file `sf_polylog.tcc`.

References `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

```
8.3.2.186  template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_pos ( unsigned int __s, std::complex<
           _Tp> __w )
```

Here s is a positive integer and the function descends into the different kernels depending on w .

Parameters

<code>__s</code>	a positive integer.
<code>__w</code>	an arbitrary complex number.

Returns

The value of the polylogarithm.

Definition at line 767 of file sf_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__fpequal()`, `__fpreal()`, `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp()`.

8.3.2.187 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_pos(unsigned int __s, _Tp __w)`

Here `s` is a positive integer and the function descends into the different kernels depending on `w`.

Parameters

<code>__s</code>	a positive integer
<code>__w</code>	an arbitrary real argument <code>w</code>

Returns

the value of the polylogarithm.

Definition at line 815 of file sf_polylog.tcc.

References `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

8.3.2.188 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg(_Tp __s, std::complex<_Tp> __w)`

This function treats the cases of negative real index `s`. Theoretical convergence is present for $|w| < 2\pi$. We use an optimized version of

$$Li_s(e^w) = \Gamma(1-s)(-w)^{(s-1)} + (2\pi)^{(s-1)}/\pi A_p(w)$$

$$A_p(w) = \sum_k \Gamma(1+k-s)/k! \sin(\pi/2 * (s-k))(w/2/\pi)^k \zeta(1+k-s)$$

Parameters

<code>__s</code>	The real index
<code>__w</code>	The complex argument

Returns

The value of the polylogarithm.

Definition at line 346 of file sf_polylog.tcc.

References `__fpequal()`, `__riemann_zeta()`, and `__riemann_zeta_m_1()`.

Referenced by `__polylog_exp_int_neg()`, and `__polylog_exp_real_neg()`.

8.3.2.189 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_neg (int __s, std::complex<_Tp> __w)`

This function treats the cases of negative integer index `s` and branches accordingly

Parameters

<code>__s</code>	the integer index s.
<code>__w</code>	The Argument w

Returns

The value of the Polylogarithm evaluated by a suitable function.

Definition at line 564 of file sf_polylog.tcc.

References `__polylog_exp_neg_even()`, and `__polylog_exp_neg_odd()`.

8.3.2.190 `template<typename _Tp, int __sigma> std::complex<_Tp> std::__detail::__polylog_exp_neg_even (unsigned int __n, std::complex<_Tp> __w)`

This function treats the cases of negative integer index s which are multiples of two.

In that case the sine occurring in the expansion occasionally takes on the value zero. We use that to provide an optimized series for $p = 2n$:

In the template parameter sigma we transport whether $p = 4k$ ($\sigma = 1$) or $p = 4k + 2$ ($\sigma = -1$)

$$Li_p(e^w) = Gamma(1-p)(-w)^{p-1} - A_p(w) - \sigma * B_p(w)$$

with

$$A_p(w) = 2(2\pi)^{(p-1)}(-p)!/(2\pi)^{(-p/2)}(1+w^2/(4\pi^2))^{-1/2+p/2} \cos((1-p)ArcTan(2\pi/w))$$

and

$$B_p(w) = -2(2\pi)^{(p-1)} \sum_{k=0}^{\infty} \Gamma(2+2k-p)/(2k+1)!(-1)^k (w/2\pi)^{(2k+1)} (\zeta(2+2k-p) - 1)$$

This is suitable for $|w| < 2\pi$ The original series is (This might be worthwhile if we use the already present table of the Bernoullis)

$$Li_p(e^w) = \Gamma(1-p)(-w)^{p-1} - \sigma(2\pi)^p/\pi \sum_{k=0}^{\infty} \Gamma(2+2k-p)/(2k+1)!(-1)^k (w/2\pi)^{(2k+1)} \zeta(2+2k-p)$$

Parameters

<code>__n</code>	the integral index $n = 4k$.
<code>__w</code>	The complex argument w

Returns

the value of the Polylogarithm.

Definition at line 450 of file sf_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp_neg()`.

8.3.2.191 `template<typename _Tp, int __sigma> std::complex<_Tp> std::__detail::__polylog_exp_neg_odd (unsigned int __n, std::complex<_Tp> __w)`

This function treats the cases of negative integer index s which are odd.

In that case the sine occuring in the expansion occasionally vanishes. We use that to provide an optimized series for $p = 1 + 2k$: In the template parameter sigma we transport whether $p = 1 + 4k$ ($\sigma = 1$) or $p = 3 + 4k$ ($\sigma = -1$)

$$Li_p(e^w) = \Gamma(1-p)(-w)^{p-1} + \sigma * A_p(w) - \sigma * B_p(w)$$

with

$$A_p(w) = 2(2\pi)^{(p-1)}\Gamma(1-p)(1+w^2/(4\pi^2))^{-1/2+p/2} \cos((1-p)ArcTan(2\pi i/w))$$

and

$$B_p(w) = 2(2\pi i)^{(p-1)} \sum_{k=0}^{\infty} \Gamma(1+2k-p)/(2k)! (-w^2/4/\pi^2)^k (\zeta(1+2k-p) - T p1)$$

This is suitable for $|w| < 2\pi$. The use of evenzeta gives a speedup of about 50 The original series is (This might be worthwhile if we use the already present table of the Bernoullis)

$$Li_p(e^w) = \Gamma(1-p) * (-w)^{p-1} - \sigma 2(2\pi)^{(p-1)} * \sum_{k=0}^{\infty} \Gamma(1+2k-p)/(2k)! (-1)^k (w/2/\pi)^{(2k)} \zeta(1+2k-p)$$

Parameters

<code>__n</code>	the integral index n = 4k.
<code>__w</code>	The complex argument w.

Returns

The value of the Polylogarithm.

Definition at line 517 of file sf_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp_neg()`.

8.3.2.192 `template<typename _PowTp, typename _Tp> _Tp std::__detail::__polylog_exp_negative_real_part (_PowTp __s, _Tp __w)`

Theoretical convergence for $\text{Re}(w) < 0$.

Seems to beat the other expansions for $\text{Re}(w) < -\pi/2 - \pi/5$. Note that this is an implementation of the basic series:

$$Li_s(e^z) = \sum_{k=1}^{\infty} e^{(k * z)} * k^{(-s)}$$

Parameters

<code>__s</code>	is an arbitrary type, integral or float.
<code>__w</code>	something with a negative real part.

Returns

the value of the polylogarithm.

Definition at line 737 of file sf_polylog.tcc.

References `__fpequal()`.

Referenced by `__polylog_exp()`, `__polylog_exp_int_neg()`, `__polylog_exp_int_pos()`, `__polylog_exp_real_neg()`, and `__polylog_exp_real_pos()`.

8.3.2.193 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos (unsigned int __s, std::complex<_Tp> __w)`

This function treats the cases of positive integer index s.

$$Li_s(e^w) = \sum_{k=0, k! \neq s-1} \zeta(s-k) w^k / k! + (H_{s-1} - \log(-w)) w^{(s-1)} / (s-1)!$$

The radius of convergence is $|w| < 2\pi i$. Note that this series involves a $\log(-x)$. gcc and Mathematica differ in their implementation of $\log(e^{i\pi})$: gcc: $\log(e^{(} + -i * \pi)) = + - i\pi$ whereas Mathematica doesn't preserve the sign in this case: $\log(e^{(} + -i\pi)) = +i\pi$

Parameters

<code>__s</code>	the index s.
<code>__w</code>	the argument w.

Returns

the value of the polylogarithm.

Definition at line 206 of file sf_polylog.tcc.

References `__fpequal()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp_int_pos()`, and `__polylog_exp_real_pos()`.

8.3.2.194 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos (unsigned int __s, _Tp __w)`

This function treats the cases of positive integer index s for real w.

This specialization is worthwhile to catch the differing behaviour of $\log(x)$.

$$Li_s(e^w) = \sum_{k=0, k! \neq s-1} \zeta(s-k) w^k / k! + (H_{s-1} - \log(-w)) w^{(s-1)} / (s-1)!$$

The radius of convergence is $|w| < 2\pi$. Note that this series involves a $\log(-x)$. The use of `evenzeta` yields a speedup of about 2.5. gcc and Mathematica differ in their implementation of $\log(e^{i\pi})$: gcc: $\log(e^{(} + -i\pi)) = + - i\pi$ whereas Mathematica doesn't preserve the sign in this case: $\log(e^{(} + -i * \pi)) = +i\pi$

Parameters

<code>__s</code>	the index.
<code>__w</code>	the argument

Returns

the value of the Polylogarithm

Definition at line 279 of file sf_polylog.tcc.

References `__fpequal()`, and `__riemann_zeta()`.

8.3.2.195 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_pos (_Tp __s, std::complex<_Tp> __w)`

This function treats the cases of positive real index s.

The defining series is

$$Li_s(e^w) = A_s(w) + B_s(w) + \Gamma(1-s)(-w)^{(s-1)}$$

with

$$A_s(w) = \sum_{k=0}^m \zeta(s-k) w^k / k!$$

$$B_s(w) = \sum_{k=m+1}^{\infty} \sin(\pi/2(s-k)) \Gamma(1-s+k) \zeta(1-s+k) (w/2/\pi)^k / k!$$

Parameters

<code>__s</code>	the positive real index s.
<code>__w</code>	The complex argument w.

Returns

the value of the polylogarithm.

Definition at line 603 of file sf_polylog.tcc.

References `__fpequal()`, and `__riemann_zeta()`.

8.3.2.196 `template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_real_neg (_Tp __s, std::complex<_Tp > __w)`

Return the polylog where s is a negative real value and for complex argument. Now we branch depending on the properties of w in the specific functions

Parameters

<code>__s</code>	A negative real value that does not reduce to a negative integer.
<code>__w</code>	The complex argument.

Returns

The value of the polylogarithm.

Definition at line 985 of file sf_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_neg_real_part()`.

Referenced by `__polylog_exp()`.

8.3.2.197 `template<typename _Tp > std::complex<_Tp> std::__detail::__polylog_exp_real_neg (_Tp __s, _Tp __w)`

Return the polylog where s is a negative real value and for real argument. Now we branch depending on the properties of w in the specific functions.

Parameters

<code>__s</code>	A negative real value.
<code>__w</code>	A real argument.

Returns

The value of the polylogarithm.

Definition at line 1013 of file sf_polylog.tcc.

References `__polylog_exp_asymp()`, `__polylog_exp_neg()`, and `__polylog_exp_negative_real_part()`.

8.3.2.198 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_pos (_Tp __s, std::complex<_Tp> __w)`

Return the polylog where s is a positive real value and for complex argument.

Parameters

<code>__s</code>	A positive real number.
<code>__w</code>	the complex argument.

Returns

The value of the polylogarithm.

Definition at line 922 of file sf_polylog.tcc.

References `__clamp_0_m2pi()`, `__clamp_pi()`, `__fpequal()`, `__fpreal()`, `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

Referenced by `__polylog_exp()`.

8.3.2.199 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_real_pos (_Tp __s, _Tp __w)`

Return the polylog where s is a positive real value and the argument is real.

Parameters

<code>__s</code>	A positive real number tht does not reduce to an integer.
<code>__w</code>	The real argument w.

Returns

The value of the polylogarithm.

Definition at line 956 of file sf_polylog.tcc.

References `__fpequal()`, `__polylog_exp_asymp()`, `__polylog_exp_negative_real_part()`, `__polylog_exp_pos()`, and `__riemann_zeta()`.

8.3.2.200 `template<typename _Tp> _Tp std::__detail::__psi (_Tp __x)`

Return the digamma function. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

For negative argument the reflection formula is used:

$$\psi(x) = \psi(1-x) - \pi \cot(\pi x)$$

.

Definition at line 2330 of file sf_gamma.tcc.

References `__psi_asymp()`.

Referenced by `__expint_En_series()`, `__hydrogen()`, `__hyperg_reflect()`, and `__psi()`.

8.3.2.201 `template<typename _Tp> _Tp std::__detail::__psi (unsigned int __n, _Tp __x)`

Return the polygamma function $\psi^{(n)}(x)$.

The polygamma function is related to the Hurwitz zeta function:

$$\psi^{(n)}(x) = (-1)^{n+1} m! \zeta(m+1, x)$$

Definition at line 2395 of file sf_gamma.tcc.

References `__hurwitz_zeta()`, `__log_gamma()`, and `__psi()`.

8.3.2.202 `template<typename _Tp> _Tp std::__detail::__psi_asymp (_Tp __x)`

Return the digamma function for large argument. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

.

The asymptotic series is given by:

$$\psi(x) = \ln(x) - \frac{1}{2x} - \sum_{n=1}^{\infty} \frac{B_{2n}}{2nx^{2n}}$$

Definition at line 2299 of file sf_gamma.tcc.

Referenced by `__psi()`.

8.3.2.203 `template<typename _Tp> _Tp std::__detail::__psi_series (_Tp __x)`

Return the digamma function by series expansion. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

.

The series is given by:

$$\psi(x) = -\gamma_E - \frac{1}{x} \sum_{k=1}^{\infty} \frac{x-1}{(k+1)(x+k)}$$

Definition at line 2268 of file sf_gamma.tcc.

8.3.2.204 `template<typename _Tp> _Tp std::__detail::__riemann_zeta (_Tp __s)`

Return the Riemann zeta function $\zeta(s)$.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} k^{-s} \text{ for } s > 1 \quad \frac{(2\pi)^s}{\pi} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s) \zeta(1-s) \text{ for } s < 1$$

For $s < 1$ use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Parameters

<code>__s</code>	The argument
------------------	--------------

Definition at line 505 of file `sf_zeta.tcc`.

References `__gamma()`, `__log_gamma()`, `__riemann_zeta_glob()`, `__riemann_zeta_product()`, and `__riemann_zeta_sum()`.

Referenced by `__polylog_exp_int_pos()`, `__polylog_exp_neg()`, `__polylog_exp_pos()`, `__polylog_exp_real_pos()`, and `evenzeta()`.

8.3.2.205 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_alt (_Tp __s)`

Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For $s < 1$ use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Definition at line 329 of file `sf_zeta.tcc`.

8.3.2.206 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_euler_maclaurin (_Tp __s)`

Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.

This is a specialization of the code for the Hurwitz zeta function.

Definition at line 282 of file `sf_zeta.tcc`.

References `_S_Euler_Maclaurin_zeta`.

8.3.2.207 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_glob (_Tp __s)`

Evaluate the Riemann zeta function by series for all $s \neq 1$. Convergence is great until largish negative numbers. Then the convergence of the > 0 sum gets better.

The series is:

$$\zeta(s) = \frac{1}{1-2^{1-s}} \sum_{n=0}^{\infty} \frac{1}{2^{n+1}} \sum_{k=0}^n (-1)^k \frac{n!}{(n-k)!k!} (k+1)^{-s}$$

Havil 2003, p. 206.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For $s < 1$ use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Definition at line 374 of file sf_zeta.tcc.

References `__log_gamma()`.

Referenced by `__riemann_zeta()`.

8.3.2.208 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_m_1 (_Tp __s)`

Return the Riemann zeta function $\zeta(s) - 1$.

Parameters

<code>__s</code>	The argument $s! = 1$
------------------	-----------------------

Definition at line 672 of file sf_zeta.tcc.

References `__riemann_zeta_m_1_sum()`, `_S_num_zetam1`, and `_S_zetam1`.

Referenced by `__polylog_exp_neg()`.

8.3.2.209 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_m_1_sum (_Tp __s)`

Return the Riemann zeta function $\zeta(s) - 1$ by summation for $s > 1$. This is a small remainder for large s .

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

Parameters

<code>__s</code>	The argument $s! = 1$
------------------	-----------------------

Definition at line 645 of file sf_zeta.tcc.

Referenced by `__riemann_zeta_m_1()`.

8.3.2.210 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_product (_Tp __s)`

Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.

$$\zeta(s) = \prod_{i=1}^{\infty} \frac{1}{1 - p_i^{-s}}$$

where p_i are the prime numbers.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For $s < 1$ use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Parameters

<code>__s</code>	The argument
------------------	--------------

Definition at line 463 of file sf_zeta.tcc.

Referenced by `__riemann_zeta()`.

8.3.2.211 `template<typename _Tp> _Tp std::__detail::__riemann_zeta_sum (_Tp __s)`

Compute the Riemann zeta function $\zeta(s)$ by summation for $s > 1$.

The Riemann zeta function is defined by:

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s} \text{ for } s > 1$$

For $s < 1$ use the reflection formula:

$$\zeta(s) = 2^s \pi^{s-1} \Gamma(1-s) \zeta(1-s)$$

Definition at line 254 of file sf_zeta.tcc.

Referenced by `__riemann_zeta()`.

8.3.2.212 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc (_Tp __a, _Tp __x)`

Return the generalized sinus cardinal function

$$\text{sinc}_a(x) = \frac{\sin(\pi x/a)}{(\pi x/a)}$$

.

Definition at line 51 of file sf_cardinal.tcc.

8.3.2.213 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc (_Tp __x)`

Return the normalized sinus cardinal function

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

.

Definition at line 98 of file sf_cardinal.tcc.

8.3.2.214 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinc_pi (_Tp __x)`

Return the unnormalized sinus cardinal function

$$\text{sinc}_{\pi}(x) = \frac{\sin(x)}{x}$$

.

Definition at line 78 of file sf_cardinal.tcc.

8.3.2.215 `template<typename _Tp> std::pair<_Tp, _Tp> std::__detail::__sincosint (_Tp __x)`

This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a `pair`.

The sine integral is defined by:

$$Si(x) = \int_0^x dt \frac{\sin(t)}{t}$$

The cosine integral is defined by:

$$Ci(x) = \gamma_E + \log(x) + \int_0^x dt \frac{\cos(t) - 1}{t}$$

Definition at line 227 of file `sf_trigint.tcc`.

References `__sincosint_asymp()`, `__sincosint_cont_frac()`, and `__sincosint_series()`.

8.3.2.216 `template<typename _Tp> void std::__detail::__sincosint_asymp (_Tp __t, _Tp & _Si, _Tp & _Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.

The asymptotic series is very good for $x > 50$.

Definition at line 163 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

8.3.2.217 `template<typename _Tp> void std::__detail::__sincosint_cont_frac (_Tp __t, _Tp & _Si, _Tp & _Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.

Definition at line 55 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

8.3.2.218 `template<typename _Tp> void std::__detail::__sincosint_series (_Tp __t, _Tp & _Si, _Tp & _Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.

Definition at line 98 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

8.3.2.219 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc (_Tp __a, _Tp __x)`

Return the generalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$

.

Definition at line 124 of file `sf_cardinal.tcc`.

8.3.2.220 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc (_Tp __x)`

Return the normalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}(x) = \frac{\sinh(\pi x)}{\pi x}$$

.

Definition at line 167 of file sf_cardinal.tcc.

8.3.2.221 `template<typename _Tp > __gnu_cxx::__promote_num_t<_Tp> std::__detail::__sinhc_pi (_Tp __x)`

Return the unnormalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_{\pi}(x) = \frac{\sinh(x)}{x}$$

.

Definition at line 149 of file sf_cardinal.tcc.

8.3.2.222 `template<typename _Tp > _Tp std::__detail::__sinhint (const _Tp __x)`

Return the hyperbolic sine integral $li(x)$.

The hyperbolic sine integral is given by

$$Shi(x) = (Ei(x) - E_1(x))/2$$

Parameters

<code>__x</code>	The argument of the hyperbolic sine integral function.
------------------	--

Returns

The hyperbolic sine integral.

Definition at line 581 of file sf_expint.tcc.

References `__expint_E1()`, and `__expint_Ei()`.

8.3.2.223 `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 695 of file sf_bessel.tcc.

References `__sph_bessel_jn()`.

8.3.2.224 `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 1314 of file sf_hankel.tcc.

References `__sph_hankel()`.

8.3.2.225 `template<typename _Tp> void std::__detail::__sph_bessel_ik (unsigned int __n, _Tp __x, _Tp & __i_n, _Tp & __k_n, _Tp & __ip_n, _Tp & __kp_n)`

Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i'_n(x)$ and $k'_n(x)$ respectively.

Parameters

<code>__n</code>	The order of the modified spherical Bessel function.
<code>__x</code>	The argument of the modified spherical Bessel function.
<code>__i_n</code>	The output regular modified spherical Bessel function.
<code>__k_n</code>	The output irregular modified spherical Bessel function.
<code>__ip_n</code>	The output derivative of the regular modified spherical Bessel function.
<code>__kp_n</code>	The output derivative of the irregular modified spherical Bessel function.

Definition at line 456 of file sf_mod_bessel.tcc.

References `__cyl_bessel_ik()`.

8.3.2.226 `template<typename _Tp> void std::__detail::__sph_bessel_jn (unsigned int __n, _Tp __x, _Tp & __j_n, _Tp & __n_n, _Tp & __jp_n, _Tp & __np_n)`

Compute the spherical Bessel $j_n(x)$ and Neumann $n_n(x)$ functions and their first derivatives $j'_n(x)$ and $n'_n(x)$ respectively.

Parameters

	<code>__n</code>	The order of the spherical Bessel function.
	<code>__x</code>	The argument of the spherical Bessel function.
out	<code>__j_n</code>	The output spherical Bessel function.
out	<code>__n_n</code>	The output spherical Neumann function.
out	<code>__jp_n</code>	The output derivative of the spherical Bessel function.
out	<code>__np_n</code>	The output derivative of the spherical Neumann function.

Definition at line 660 of file sf_bessel.tcc.

References `__cyl_bessel_jn()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

```
8.3.2.227 template<typename _Tp > void std::__detail::__sph_hankel ( unsigned int __n, std::complex< _Tp > __z,  
    std::complex< _Tp > &_H1, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2, std::complex< _Tp > &_H2p  
    )
```

Helper to compute complex spherical Hankel functions and their derivatives.

Parameters

in	<code>__n</code>	The order for which the spherical Hankel functions are evaluated.
in	<code>__z</code>	The argument at which the spherical Hankel functions are evaluated.
out	<code>_H1</code>	The spherical Hankel function of the first kind.
out	<code>_H1p</code>	The derivative of the spherical Hankel function of the first kind.
out	<code>_H2</code>	The spherical Hankel function of the second kind.
out	<code>_H2p</code>	The derivative of the spherical Hankel function of the second kind.

Definition at line 1258 of file sf_hankel.tcc.

References `__hankel()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

8.3.2.228 `template<typename _Tp> std::complex<_Tp> std::__detail::__sph_hankel_1 (unsigned int __n, _Tp __x)`

Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.

The spherical Hankel function of the first kind is defined by:

$$h_n^{(1)}(x) = j_n(x) + in_n(x)$$

Parameters

<code>__n</code>	The order of the spherical Neumann function.
<code>__x</code>	The argument of the spherical Neumann function.

Returns

The output spherical Neumann function.

Definition at line 764 of file sf_bessel.tcc.

References `__sph_bessel_jn()`.

8.3.2.229 `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	<code>__n</code>	The order for which the spherical Hankel function of the first kind is evaluated.
in	<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 1282 of file sf_hankel.tcc.

References `__sph_hankel()`.

8.3.2.230 `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

<code>__n</code>	The non-negative integral order
<code>__x</code>	The non-negative real argument

Returns

The output spherical Neumann function.

Definition at line 797 of file `sf_bessel.tcc`.

References `__sph_bessel_jn()`.

8.3.2.231 `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

<code>in</code>	<code>__n</code>	The order for which the spherical Hankel function of the second kind is evaluated.
<code>in</code>	<code>__z</code>	The argument at which the spherical Hankel function of the second kind is evaluated.

Returns

The complex spherical Hankel function of the second kind.

Definition at line 1298 of file `sf_hankel.tcc`.

References `__sph_hankel()`.

8.3.2.232 `template<typename _Tp> std::complex<_Tp> std::__detail::__sph_harmonic (unsigned int __l, int __m, _Tp __theta, _Tp __phi)`

Return the spherical harmonic function.

The spherical harmonic function of l , m , and θ , ϕ is defined by:

$$Y_l^m(\theta, \phi) = (-1)^m \left[\frac{(2l+1)}{4\pi} \frac{(l-m)!}{(l+m)!} \right] P_l^{|m|}(\cos \theta) \exp^{im\phi}$$

Parameters

<code>__l</code>	The order of the spherical harmonic function. $l \geq 0$.
<code>__m</code>	The order of the spherical harmonic function. $m \leq l$.
<code>__theta</code>	The radian polar angle argument of the spherical harmonic function.
<code>__phi</code>	The radian azimuthal angle argument of the spherical harmonic function.

Definition at line 350 of file sf_legendre.tcc.

References `__sph_legendre()`.

8.3.2.233 `template<typename _Tp> _Tp std::__detail::__sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`

Return the spherical associated Legendre function.

The spherical associated Legendre function of l , m , and θ is defined as $Y_l^m(\theta, 0)$ where

$$Y_l^m(\theta, \phi) = (-1)^m \left[\frac{(2l+1)}{4\pi} \frac{(l-m)!}{(l+m)!} \right] P_l^m(\cos \theta) \exp^{im\phi}$$

is the spherical harmonic function and $P_l^m(x)$ is the associated Legendre function.

This function differs from the associated Legendre function by argument ($x = \cos(\theta)$) and by a normalization factor but this factor is rather large for large l and m and so this function is stable for larger differences of l and m .

Parameters

<code>__l</code>	The order of the spherical associated Legendre function. $l \geq 0$.
<code>__m</code>	The order of the spherical associated Legendre function. $m \leq l$.
<code>__theta</code>	The radian polar angle argument of the spherical associated Legendre function.

Definition at line 253 of file sf_legendre.tcc.

References `__log_gamma()`, and `__poly_legendre_p()`.

Referenced by `__hydrogen()`, and `__sph_harmonic()`.

8.3.2.234 `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 732 of file sf_bessel.tcc.

References `__sph_bessel_jn()`.

8.3.2.235 `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	<code>__n</code>	The order for which the spherical Neumann function is evaluated.
in	<code>__z</code>	The argument at which the spherical Neumann function is evaluated.

Returns

The complex spherical Neumann function.

Definition at line 1330 of file sf_hankel.tcc.

References `__sph_hankel()`.

8.3.2.236 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdf (_Tp __t, unsigned int __nu)`

Return the Students T probability function.

The students T propability function is related to the incomplete beta function:

$$A(t|\nu) = 1 - I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right)A(t|\nu) =$$

Parameters

	<code>__t</code>	
	<code>__nu</code>	

Definition at line 301 of file sf_beta.tcc.

References `__beta_inc()`.

8.3.2.237 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdfc (_Tp __t, unsigned int __nu)`

Return the complement of the Students T probability function.

The complement of the students T propability function is:

$$A_c(t|\nu) = I_{\frac{\nu}{\nu+t^2}}\left(\frac{\nu}{2}, \frac{1}{2}\right) = 1 - A(t|\nu)$$

Parameters

	<code>__t</code>	
	<code>__nu</code>	

Definition at line 324 of file sf_beta.tcc.

References `__beta_inc()`.

8.3.2.238 `template<typename _Tp > _Tp std::__detail::__theta_1 (_Tp __nu, _Tp __x)`

Return the exponential theta-1 function of period `nu` and argument `x`.

The Neville theta-1 function is defined by

$$\theta_1(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j - 1/2)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 190 of file `sf_theta.tcc`.

References `__theta_2()`.

Referenced by `__theta_s()`.

8.3.2.239 `template<typename _Tp> _Tp std::__detail::__theta_2 (_Tp __nu, _Tp __x)`

Return the exponential theta-2 function of period `nu` and argument `x`.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 162 of file `sf_theta.tcc`.

References `__theta_2_asymp()`, and `__theta_2_sum()`.

Referenced by `__theta_1()`, and `__theta_c()`.

8.3.2.240 `template<typename _Tp> _Tp std::__detail::__theta_2_asymp (_Tp __nu, _Tp __x)`

Compute and return the θ_2 function by series expansion.

Definition at line 103 of file `sf_theta.tcc`.

Referenced by `__theta_2()`.

8.3.2.241 `template<typename _Tp> _Tp std::__detail::__theta_2_sum (_Tp __nu, _Tp __x)`

Compute and return the θ_1 function by series expansion.

Definition at line 49 of file `sf_theta.tcc`.

Referenced by `__theta_2()`.

8.3.2.242 `template<typename _Tp> _Tp std::__detail::__theta_3 (_Tp __nu, _Tp __x)`

Return the exponential theta-3 function of period `nu` and argument `x`.

The exponential theta-3 function is defined by

$$\theta_3(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 1) argument
<code>__x</code>	The argument

Definition at line 216 of file sf_theta.tcc.

References `__theta_3_asymp()`, and `__theta_3_sum()`.

Referenced by `__theta_4()`, and `__theta_d()`.

8.3.2.243 `template<typename _Tp> _Tp std::__detail::__theta_3_asymp (_Tp __nu, _Tp __x)`

Compute and return the θ_3 function by asymptotic series expansion.

Definition at line 128 of file sf_theta.tcc.

Referenced by `__theta_3()`.

8.3.2.244 `template<typename _Tp> _Tp std::__detail::__theta_3_sum (_Tp __nu, _Tp __x)`

Compute and return the θ_3 function by series expansion.

Definition at line 77 of file sf_theta.tcc.

Referenced by `__theta_3()`.

8.3.2.245 `template<typename _Tp> _Tp std::__detail::__theta_4 (_Tp __nu, _Tp __x)`

Return the exponential theta-2 function of period `nu` and argument `x`.

The exponential theta-2 function is defined by

$$\theta_2(\nu, x) = \frac{1}{\sqrt{\pi x}} \sum_{j=-\infty}^{+\infty} (-1)^j \exp\left(\frac{-(\nu + j)^2}{x}\right)$$

Parameters

<code>__nu</code>	The periodic (period = 2) argument
<code>__x</code>	The argument

Definition at line 244 of file sf_theta.tcc.

References `__theta_3()`.

Referenced by `__theta_n()`.

8.3.2.246 `template<typename _Tp> _Tp std::__detail::__theta_c (_Tp __k, _Tp __x)`

Return the Neville θ_c function

Definition at line 337 of file sf_theta.tcc.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_2()`.

8.3.2.247 `template<typename _Tp> _Tp std::__detail::__theta_d (_Tp __k, _Tp __x)`

Return the Neville θ_d function

Definition at line 362 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_3()`.

8.3.2.248 `template<typename _Tp> _Tp std::__detail::__theta_n (_Tp __k, _Tp __x)`

Return the Neville θ_n function

Definition at line 387 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_4()`.

8.3.2.249 `template<typename _Tp> _Tp std::__detail::__theta_s (_Tp __k, _Tp __x)`

Return the Neville θ_s function

Definition at line 311 of file `sf_theta.tcc`.

References `__comp_ellint_1()`, `__ellnome()`, and `__theta_1()`.

8.3.2.250 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)`

Return the Zernicke polynomial $Z_n^m(\rho, \phi)$ for non-negative degree n , signed order m , and real radial argument ρ and azimuthal angle ϕ .

The even Zernicke polynomials are defined by:

$$Z_n^m(\rho, \phi) = R_n^m(\rho) \cos(m\phi)$$

and the odd Zernicke polynomials are defined by:

$$Z_n^{-m}(\rho, \phi) = R_n^m(\rho) \sin(m\phi)$$

for non-negative degree m and $m \leq n$ and where $R_n^m(\rho)$ is the radial polynomial (

See also

[__poly_radial_jacobi](#)).

Template Parameters

<code>_Tp</code>	The real type of the radial coordinate and azimuthal angle
------------------	--

Parameters

<code>__n</code>	The non-negative degree.
<code>__m</code>	The azimuthal order

<code>__rho</code>	The radial coordinate
<code>__phi</code>	The azimuthal angle

Definition at line 183 of file sf_jacobi.tcc.

References `__poly_radial_jacobi()`.

8.3.2.251 `template<typename _Tp > _Tp std::__detail::__znorm1 (_Tp __x)`

Definition at line 58 of file sf_owens_t.tcc.

Referenced by `__owens_t()`.

8.3.2.252 `template<typename _Tp > _Tp std::__detail::__znorm2 (_Tp __x)`

Definition at line 47 of file sf_owens_t.tcc.

Referenced by `__owens_t()`.

8.3.2.253 `template<typename _Tp = double> _Tp std::__detail::evenzeta (unsigned int __k)`

A function to calculate the values of zeta at even positive integers. For values smaller than thirty a table is used.

Parameters

<code>__k</code>	an integer at which we evaluate the Riemann zeta function.
------------------	--

Returns

$zeta(k)$

Definition at line 156 of file sf_polylog.tcc.

References `__riemann_zeta()`.

8.3.3 Variable Documentation

8.3.3.1 `constexpr size_t std::__detail::_Num_Euler_Maclaurin_zeta = 100`

Coefficients for Euler-Maclaurin summation of zeta functions.

$$B_{2j}/(2j)!$$

where B_k are the Bernoulli numbers.

Definition at line 65 of file sf_zeta.tcc.

8.3.3.2 `constexpr _Factorial_table<long double> std::__detail::_S_double_factorial_table[301]`

Definition at line 274 of file sf_gamma.tcc.

Referenced by `__double_factorial()`, and `__log_double_factorial()`.

8.3.3.3 constexpr long double std::__detail::_S_Euler_Maclaurin_zeta[Num_Euler_Maclaurin_zeta]

Definition at line 68 of file sf_zeta.tcc.

Referenced by __hurwitz_zeta_euler_maclaurin(), and __riemann_zeta_euler_maclaurin().

8.3.3.4 constexpr _Factorial_table<long double> std::__detail::_S_factorial_table[171]

Definition at line 84 of file sf_gamma.tcc.

Referenced by __factorial(), and __log_factorial().

8.3.3.5 constexpr _Factorial_table<long double> std::__detail::_S_neg_double_factorial_table[999]

Definition at line 595 of file sf_gamma.tcc.

Referenced by __double_factorial(), and __log_double_factorial().

8.3.3.6 template<typename _Tp > constexpr std::size_t std::__detail::_S_num_double_factorials = 0

Definition at line 259 of file sf_gamma.tcc.

8.3.3.7 template<> constexpr std::size_t std::__detail::_S_num_double_factorials< double > = 301

Definition at line 264 of file sf_gamma.tcc.

8.3.3.8 template<> constexpr std::size_t std::__detail::_S_num_double_factorials< float > = 57

Definition at line 262 of file sf_gamma.tcc.

8.3.3.9 template<> constexpr std::size_t std::__detail::_S_num_double_factorials< long double > = 301

Definition at line 266 of file sf_gamma.tcc.

8.3.3.10 template<typename _Tp > constexpr std::size_t std::__detail::_S_num_factorials = 0

Definition at line 69 of file sf_gamma.tcc.

8.3.3.11 template<> constexpr std::size_t std::__detail::_S_num_factorials< double > = 171

Definition at line 74 of file sf_gamma.tcc.

8.3.3.12 template<> constexpr std::size_t std::__detail::_S_num_factorials< float > = 35

Definition at line 72 of file sf_gamma.tcc.

8.3.3.13 `template<> constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171`

Definition at line 76 of file `sf_gamma.tcc`.

8.3.3.14 `template<typename _Tp > constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0`

Definition at line 579 of file `sf_gamma.tcc`.

8.3.3.15 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150`

Definition at line 584 of file `sf_gamma.tcc`.

8.3.3.16 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27`

Definition at line 582 of file `sf_gamma.tcc`.

8.3.3.17 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999`

Definition at line 586 of file `sf_gamma.tcc`.

8.3.3.18 `constexpr size_t std::__detail::_S_num_zetam1 = 33`

Table of $\zeta(n) - 1$ from 2 - 32. MPFR - 128 bits.

Definition at line 592 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

8.3.3.19 `constexpr long double std::__detail::_S_zetam1[_S_num_zetam1]`

Definition at line 596 of file `sf_zeta.tcc`.

Referenced by `__riemann_zeta_m_1()`.

Chapter 9

Class Documentation

9.1 `std::__detail::_Factorial_table<_Tp>` Struct Template Reference

Public Attributes

- `_Tp __factorial`
- `_Tp __log_factorial`
- `unsigned int __n`

9.1.1 Detailed Description

```
template<typename _Tp>struct std::__detail::_Factorial_table<_Tp>
```

Definition at line 61 of file `sf_gamma.tcc`.

9.1.2 Member Data Documentation

9.1.2.1 `template<typename _Tp>_Tp std::__detail::_Factorial_table<_Tp>::__factorial`

Definition at line 64 of file `sf_gamma.tcc`.

9.1.2.2 `template<typename _Tp>_Tp std::__detail::_Factorial_table<_Tp>::__log_factorial`

Definition at line 65 of file `sf_gamma.tcc`.

9.1.2.3 `template<typename _Tp> unsigned int std::__detail::_Factorial_table<_Tp>::__n`

Definition at line 63 of file `sf_gamma.tcc`.

The documentation for this struct was generated from the following file:

- `bits/sf_gamma.tcc`

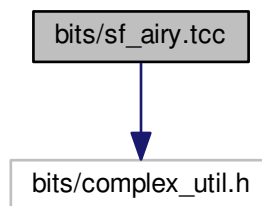
Chapter 10

File Documentation

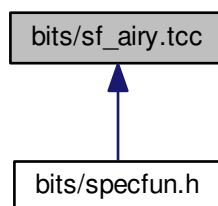
10.1 bits/sf_airy.tcc File Reference

```
#include <bits/complex_util.h>
```

Include dependency graph for sf_airy.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_AIRY_TCC 1](#)

Functions

- `template<typename _Tp >`
`void std::__detail::__airy (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)`
This function computes the Airy function $Ai(z)$ and its first derivative in the complex z -plane.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__airy_ai (std::complex< _Tp > __z)`
Return the complex Airy Ai function.
- `template<typename _Tp >`
`void std::__detail::__airy_asymp_absarg_ge_pio3 (std::complex< _Tp > __z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, int __sign=-1)`
*This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(z)| < 2 * \pi/3$. For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from $abs(z)$.*
- `template<typename _Tp >`
`void std::__detail::__airy_asymp_absarg_lt_pio3 (std::complex< _Tp > __z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip)`
This function evaluates $Ai(z)$ and $Ai'(z)$ from their asymptotic expansions for $|arg(-z)| < \pi/3$. For speed, the number of terms needed to achieve about 16 decimals accuracy is tabled and determined from $|z|$.
- `template<typename _Tp >`
`void std::__detail::__airy_bessel_i (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_lp1d3, std::complex< _Tp > &_lm1d3, std::complex< _Tp > &_lp2d3, std::complex< _Tp > &_lm2d3)`
- `template<typename _Tp >`
`void std::__detail::__airy_bessel_k (const std::complex< _Tp > &__z, _Tp __eps, std::complex< _Tp > &_Kp1d3, std::complex< _Tp > &_Kp2d3)`
Compute approximations to the modified Bessel functions of the second kind of orders 1/3 and 2/3 for moderate arguments.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__airy_bi (std::complex< _Tp > __z)`
Return the complex Airy Bi function.
- `template<typename _Tp >`
`void std::__detail::__airy_hyperg_rational (const std::complex< _Tp > &__z, std::complex< _Tp > &_Ai, std::complex< _Tp > &_Aip, std::complex< _Tp > &_Bi, std::complex< _Tp > &_Bip)`
This function computes rational approximations to the hypergeometric functions related to the modified Bessel functions of orders $\nu = + - 1/3$ and $\nu = -2/3$. That is, $A(z)/B(z)$, Where $A(z)$ and $B(z)$ are cubic polynomials with real coefficients, approximates

$$\frac{\Gamma(\nu + 1)}{(z/2)^{\nu}} I_{\nu}(z) = {}_0F_1(; \nu + 1; z^2/4),$$

where the function on the right is a confluent hypergeometric limit function. For $|z| \leq 1/4$ and $|arg(z)| \leq \pi/2$, the approximations are accurate to about 16 decimals.

10.1.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

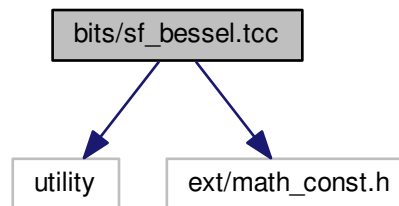
10.1.2 Macro Definition Documentation

10.1.2.1 `#define _GLIBCXX_BITS_SF_AIRY_TCC 1`

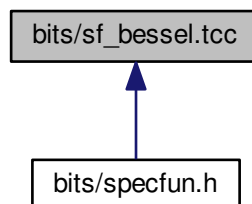
Definition at line 31 of file `sf_airy.tcc`.

10.2 bits/sf_bessel.tcc File Reference

```
#include <utility>
#include <ext/math_const.h>
Include dependency graph for sf_bessel.tcc:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)

- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_BESSEL_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__cyl_bessel_ij_series (_Tp __nu, _Tp __x, _Tp __sgn, unsigned int __max_iter)`
This routine returns the cylindrical Bessel functions of order ν : J_ν or I_ν by series expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__cyl_bessel_j (_Tp __nu, _Tp __x)`
Return the Bessel function of order ν : $J_\nu(x)$.
- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_jn (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
Return the cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_jn_asymp (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
This routine computes the asymptotic cylindrical Bessel and Neumann functions of order ν : $J_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.
- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_jn_steel (_Tp __nu, _Tp __x, _Tp &_Jnu, _Tp &_Nnu, _Tp &_Jpnu, _Tp &_Npnu)`
Compute the Bessel $J_\nu(x)$ and Neumann $N_\nu(x)$ functions and their first derivatives $J'_\nu(x)$ and $N'_\nu(x)$ respectively. These four functions are computed together for numerical stability.
- `template<typename _Tp >`
`std::complex<_Tp> std::__detail::__cyl_hankel_1 (_Tp __nu, _Tp __x)`
Return the cylindrical Hankel function of the first kind $H_\nu^{(1)}(x)$.
- `template<typename _Tp >`
`std::complex<_Tp> std::__detail::__cyl_hankel_2 (_Tp __nu, _Tp __x)`
Return the cylindrical Hankel function of the second kind $H_n^{(2)}u(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__cyl_neumann_n (_Tp __nu, _Tp __x)`
Return the Neumann function of order ν : $N_\nu(x)$.
- `template<typename _Tp >`
`void std::__detail::__gamma_temme (_Tp __mu, _Tp &__gam1, _Tp &__gam2, _Tp &__gampl, _Tp &__gammi)`
Compute the gamma functions required by the Temme series expansions of $N_\nu(x)$ and $K_\nu(x)$.

$$\Gamma_1 = \frac{1}{2\mu} \left[\frac{1}{\Gamma(1-\mu)} - \frac{1}{\Gamma(1+\mu)} \right]$$

and

$$\Gamma_2 = \frac{1}{2} \left[\frac{1}{\Gamma(1-\mu)} + \frac{1}{\Gamma(1+\mu)} \right]$$

where $-1/2 \leq \mu \leq 1/2$ is $\mu = \nu - N$ and N is the nearest integer to ν . The values of $\Gamma(1+\mu)$ and $\Gamma(1-\mu)$ are returned as well.
- `template<typename _Tp >`
`_Tp std::__detail::__sph_bessel (unsigned int __n, _Tp __x)`
Return the spherical Bessel function $j_n(x)$ of order n and non-negative real argument x .

- `template<typename _Tp >`
`void std::__detail::__sph_bessel_jn (unsigned int __n, _Tp __x, _Tp &__j_n, _Tp &__n_n, _Tp &__jp_n, _Tp &__np_n)`
Compute the spherical Bessel $j_n(x)$ and Neumann $n_n(x)$ functions and their first derivatives $j'_n(x)$ and $n'_n(x)$ respectively.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, _Tp __x)`
Return the spherical Hankel function of the first kind $h_n^{(1)}(x)$.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_2 (unsigned int __n, _Tp __x)`
Return the spherical Hankel function of the second kind $h_n^{(2)}(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__sph_neumann (unsigned int __n, _Tp __x)`
Return the spherical Neumann function $n_n(x)$ of order n and non-negative real argument x .

10.2.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.2.2 Macro Definition Documentation

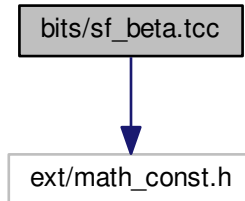
10.2.2.1 `#define _GLIBCXX_BITS_SF_BESSEL_TCC 1`

Definition at line 47 of file `sf_bessel.tcc`.

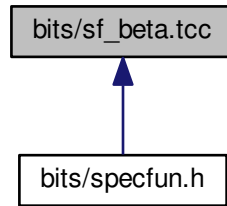
10.3 bits/sf_beta.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_beta.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_BETA_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__beta \(_Tp __a, _Tp __b\)](#)
Return the beta function $B(a, b)$.
- [template<typename _Tp > _Tp std::__detail::__beta_gamma \(_Tp __a, _Tp __b\)](#)
Return the beta function: $B(a, b)$.
- [template<typename _Tp > _Tp std::__detail::__beta_inc \(_Tp __a, _Tp __b, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__beta_inc_cont_frac \(_Tp __a, _Tp __b, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__beta_lgamma \(_Tp __a, _Tp __b\)](#)
Return the beta function $B(a, b)$ using the log gamma functions.
- [template<typename _Tp > _Tp std::__detail::__beta_product \(_Tp __a, _Tp __b\)](#)
Return the beta function $B(x, y)$ using the product form.
- [template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdf \(_Tp __p, unsigned int __n, unsigned int __k\)](#)
Return the binomial cumulative distribution function.
- [template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__binomial_cdfc \(_Tp __p, unsigned int __n, unsigned int __k\)](#)
Return the complementary binomial cumulative distribution function.

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdf (_Tp __F, unsigned int __nu1, unsigned int __nu2)`
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__fisher_f_cdfc (_Tp __F, unsigned int __nu1, unsigned int __nu2)`
Return the F-distribution propability function. This returns the probability that the observed chi-square for a correct model exceeds the value χ^2 .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdf (_Tp __t, unsigned int __nu)`
Return the Students T probability function.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__student_t_cdfc (_Tp __t, unsigned int __nu)`
Return the complement of the Students T probability function.

10.3.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.3.2 Macro Definition Documentation

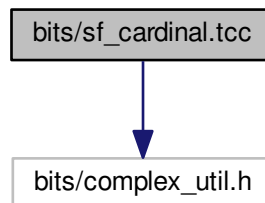
10.3.2.1 `#define _GLIBCXX_BITS_SF_BETA_TCC 1`

Definition at line 49 of file `sf_beta.tcc`.

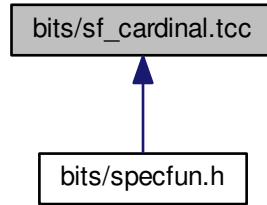
10.4 bits/sf_cardinal.tcc File Reference

```
#include <bits/complex_util.h>
```

Include dependency graph for `sf_cardinal.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_CARDINAL_TCC 1](#)

Functions

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t<_Tp > std::__detail::__sinc (_Tp __a, _Tp __x)`
Return the generalized sinus cardinal function

$$\text{sinc}_a(x) = \frac{\sin(\pi x/a)}{(\pi x/a)}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t<_Tp > std::__detail::__sinc (_Tp __x)`
Return the normalized sinus cardinal function

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t<_Tp > std::__detail::__sinc_pi (_Tp __x)`
Return the unnormalized sinus cardinal function

$$\text{sinc}_\pi(x) = \frac{\sin(x)}{x}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t<_Tp > std::__detail::__sinhc (_Tp __a, _Tp __x)`

Return the generalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinhc (_Tp __x)`

Return the normalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}(x) = \frac{\sinh(\pi x)}{\pi x}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > std::__detail::__sinhc_pi (_Tp __x)`

Return the unnormalized hyperbolic sinus cardinal function

$$\operatorname{sinhc}_\pi(x) = \frac{\sinh(x)}{x}$$

10.4.1 Macro Definition Documentation

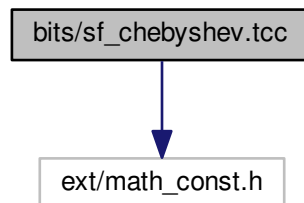
10.4.1.1 `#define _GLIBCXX_BITS_SF_CARDINAL_TCC 1`

Definition at line 30 of file `sf_cardinal.tcc`.

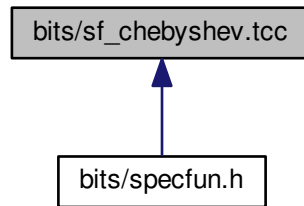
10.5 bits/sf_chebyshev.tcc File Reference

`#include <ext/math_const.h>`

Include dependency graph for `sf_chebyshev.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_SF_CHEBYSHEV_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__chebyshev_recur \(unsigned int __n, _Tp __x, _Tp _C0, _Tp _C1\)](#)
- [template<typename _Tp > _Tp std::__detail::__chebyshev_t \(unsigned int __n, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__chebyshev_u \(unsigned int __n, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__chebyshev_v \(unsigned int __n, _Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__chebyshev_w \(unsigned int __n, _Tp __x\)](#)

10.5.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.5.2 Macro Definition Documentation

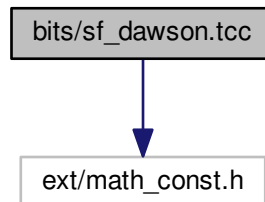
10.5.2.1 [#define _GLIBCXX_SF_CHEBYSHEV_TCC 1](#)

Definition at line 31 of file `sf_chebyshev.tcc`.

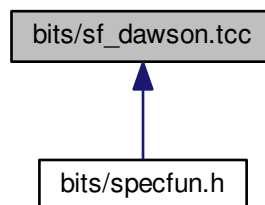
10.6 bits/sf_dawson.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_dawson.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_SF_DAWSON_TCC 1`

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__dawson (_Tp __x)`
Return the Dawson integral, $F(x)$, for real argument x .

- `template<typename _Tp >`
`_Tp std::__detail::__dawson_cont_frac (_Tp __x)`

Compute the Dawson integral using a sampling theorem representation.

- `template<typename _Tp >`
`_Tp std::__detail::__dawson_series (_Tp __x)`

Compute the Dawson integral using the series expansion.

10.6.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.6.2 Macro Definition Documentation

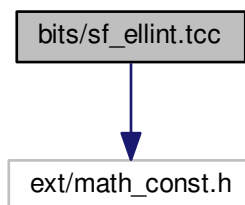
10.6.2.1 `#define _GLIBCXX_SF_DAWSON_TCC 1`

Definition at line 31 of file `sf_dawson.tcc`.

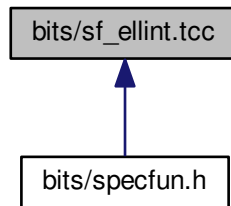
10.7 `bits/sf_ellint.tcc` File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_ellint.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_ELLINT_TCC 1](#)

Functions

- `template<typename _Tp >
_Tp std::__detail::__comp_ellint_1 (_Tp __k)`
Return the complete elliptic integral of the first kind $K(k)$ using the Carlson formulation.
- `template<typename _Tp >
_Tp std::__detail::__comp_ellint_2 (_Tp __k)`
Return the complete elliptic integral of the second kind $E(k)$ using the Carlson formulation.
- `template<typename _Tp >
_Tp std::__detail::__comp_ellint_3 (_Tp __k, _Tp __nu)`
Return the complete elliptic integral of the third kind $\Pi(k, \nu) = \Pi(k, \nu, \pi/2)$ using the Carlson formulation.
- `template<typename _Tp >
_Tp std::__detail::__comp_ellint_d (_Tp __k)`
- `template<typename _Tp >
_Tp std::__detail::__comp_ellint_rf (_Tp __x, _Tp __y)`
- `template<typename _Tp >
_Tp std::__detail::__comp_ellint_rg (_Tp __x, _Tp __y)`
- `template<typename _Tp >
_Tp std::__detail::__ellint_1 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.
- `template<typename _Tp >
_Tp std::__detail::__ellint_2 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the second kind $E(k, \phi)$ using the Carlson formulation.

- `template<typename _Tp >`
`_Tp std::__detail::__ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)`
Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_d (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_el1 (_Tp __x, _Tp __k_c)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_rc (_Tp __x, _Tp __y)`
Return the Carlson elliptic function $R_C(x, y) = R_F(x, y, y)$ where $R_F(x, y, z)$ is the Carlson elliptic function of the first kind.
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_rd (_Tp __x, _Tp __y, _Tp __z)`
Return the Carlson elliptic function of the second kind $R_D(x, y, z) = R_J(x, y, z, z)$ where $R_J(x, y, z, p)$ is the Carlson elliptic function of the third kind.
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_rf (_Tp __x, _Tp __y, _Tp __z)`
Return the Carlson elliptic function $R_F(x, y, z)$ of the first kind.
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_rg (_Tp __x, _Tp __y, _Tp __z)`
Return the symmetric Carlson elliptic function of the second kind $R_G(x, y, z)$.
- `template<typename _Tp >`
`_Tp std::__detail::__ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)`
Return the Carlson elliptic function $R_J(x, y, z, p)$ of the third kind.
- `template<typename _Tp >`
`_Tp std::__detail::__heuman_lambda (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`
`_Tp std::__detail::__jacobi_zeta (_Tp __k, _Tp __phi)`

10.7.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.7.2 Macro Definition Documentation

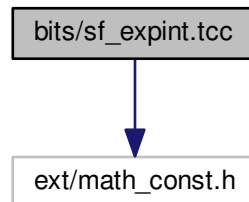
10.7.2.1 `#define _GLIBCXX_BITS_SF_ELLINT_TCC 1`

Definition at line 47 of file `sf_ellint.tcc`.

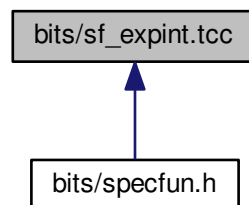
10.8 bits/sf_expint.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_expint.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_EXPINT_TCC 1](#)

Functions

- [template<typename _Tp >
_Tp std::__detail::__coshint \(const _Tp __x\)](#)
Return the hyperbolic cosine integral $li(x)$.

- `template<typename _Tp >`
`_Tp std::__detail::__expint (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint (_Tp __x)`
Return the exponential integral $Ei(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_asymp (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ for large argument.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_E1 (_Tp __x)`
Return the exponential integral $E_1(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_E1_asymp (_Tp __x)`
Return the exponential integral $E_1(x)$ by asymptotic expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_E1_series (_Tp __x)`
Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei (_Tp __x)`
Return the exponential integral $Ei(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei_asymp (_Tp __x)`
Return the exponential integral $Ei(x)$ by asymptotic expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_Ei_series (_Tp __x)`
Return the exponential integral $Ei(x)$ by series summation.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_cont_frac (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ by continued fractions.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_recursion (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_En_series (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ by series summation.
- `template<typename _Tp >`
`_Tp std::__detail::__expint_large_n (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ for large order.
- `template<typename _Tp >`
`_Tp std::__detail::__logint (const _Tp __x)`
Return the logarithmic integral $li(x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__sinhint (const _Tp __x)`
Return the hyperbolic sine integral $li(x)$.

10.8.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.8.2 Macro Definition Documentation

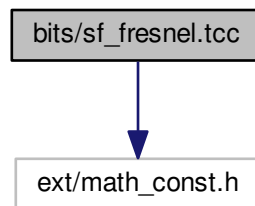
10.8.2.1 `#define _GLIBCXX_BITS_SF_EXPINT_TCC 1`

Definition at line 47 of file `sf_expint.tcc`.

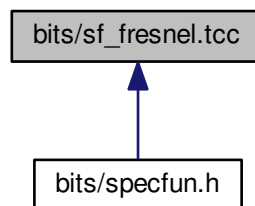
10.9 bits/sf_fresnel.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_fresnel.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)

- [std::__detail](#)

Macros

- [#define _GLIBCXX_SF_FRESNEL_TCC](#) 1

Functions

- `template<typename _Tp >
std::complex< _Tp > std::__detail::__fresnel (const _Tp __x)`
Return the Fresnel cosine and sine integrals as a complex number $\text{ff}[C(x) + iS(x)]$.
- `template<typename _Tp >
void std::__detail::__fresnel_cont_frac (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)`
This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.
- `template<typename _Tp >
void std::__detail::__fresnel_series (const _Tp __ax, _Tp &_Cf, _Tp &_Sf)`
This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.

10.9.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.9.2 Macro Definition Documentation

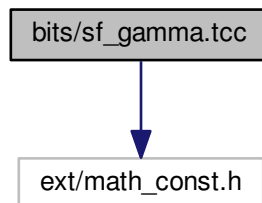
10.9.2.1 [#define _GLIBCXX_SF_FRESNEL_TCC](#) 1

Definition at line 31 of file `sf_fresnel.tcc`.

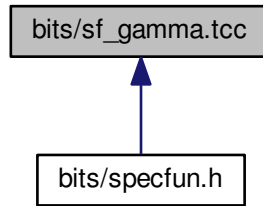
10.10 `bits/sf_gamma.tcc` File Reference

`#include <ext/math_const.h>`

Include dependency graph for `sf_gamma.tcc`:



This graph shows which files directly or indirectly include this file:



Classes

- struct `std::__detail::_Factorial_table< _Tp >`

Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_BITS_SF_GAMMA_TCC 1`

Functions

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli (int __n)`
This returns Bernoulli number B_n .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_2n (int __n)`
This returns Bernoulli number B_n .
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__bernoulli_series (unsigned int __n)`
This returns Bernoulli numbers from a table or by summation for larger values.
- `template<typename _Tp >`
`_Tp std::__detail::__bincoef (unsigned int __n, unsigned int __k)`
Return the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__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 >`
`_GLIBCXX14_CONSTEXPR _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 >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__double_factorial (int __n)`

Return the double factorial of the integer n .

- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (unsigned int __n)`

Return the factorial of the integer n .

- `template<typename _Tp >`
`_Tp std::__detail::__gamma (_Tp __x)`

Return $\Gamma(x)$.

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > std::__detail::__gamma_cont_frac (_Tp __a, _Tp __x)`

- `template<typename _Tp >`
`_Tp std::__detail::__gamma_l (_Tp __a, _Tp __x)`

Return the lower incomplete gamma function. The lower incomplete gamma function is defined by

$$\gamma(a, x) = \int_0^x e^{-t} t^{a-1} dt (a > 0)$$

- `template<typename _Tp >`
`_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 >`
`std::pair< _Tp, _Tp > std::__detail::__gamma_series (_Tp __a, _Tp __x)`

- `template<typename _Tp >`
`_Tp std::__detail::__gamma_u (_Tp __a, _Tp __x)`

Return the upper incomplete gamma function. The lower incomplete gamma function is defined by

$$\Gamma(a, x) = \int_x^\infty e^{-t} t^{a-1} dt (a > 0)$$

- template<typename _Tp >
_Tp std::__detail::__log_bincoef (unsigned int __n, unsigned int __k)

Return the logarithm of the binomial coefficient. The binomial coefficient is given by:

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (_Tp __x)
- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_double_factorial (int __n)

Return the logarithm of the double factorial of the integer n .

- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_factorial (unsigned int __n)

Return the logarithm of the factorial of the integer n .

- template<typename _Tp >
_Tp std::__detail::__log_gamma (_Tp __x)

Return $\log(|\Gamma(x)|)$. This will return values even for $x < 0$. To recover the sign of $\Gamma(x)$ for any argument use `__log_gamma_sign`.

- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_bernoulli (_Tp __x)

Return $\log(\Gamma(x))$ by asymptotic expansion with Bernoulli number coefficients. This is like Sterling's approximation.

- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_lanczos (_Tp __x)

Return $\log(\Gamma(x))$ by the Lanczos method. This method dominates all others on the positive axis I think.

- template<typename _Tp >
_Tp std::__detail::__log_gamma_sign (_Tp __x)

Return the sign of $\Gamma(x)$. At nonpositive integers zero is returned.

- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp std::__detail::__log_gamma_spouge (_Tp __z)

Return $\Gamma(z)$ by the Spouge algorithm:

$$\Gamma(z+1) = (z+a)^{z+1/2} e^{-z-a} \left[\sqrt{2\pi} \sum_{k=1}^{\lceil a \rceil + 1} \frac{c_k(a)}{z+k} \right]$$

where

$$c_k(a) = \frac{(-1)^{k-1}}{(k-1)!} (a-k)^{k-1/2} e^{a-k}$$

and the error is bounded by

$$\epsilon(a) < a^{-1/2} (2\pi)^{-a-1/2}$$

- template<typename _Tp >
_Tp std::__detail::__log_pochhammer_l (_Tp __a, _Tp __n)

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular, $\text{ff}(n)_n = n!$. Thus this function returns

$$\ln[(a)_n] = \Gamma(a+1) - \Gamma(a-n+1), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left\{ \begin{matrix} a \\ n \end{matrix} \right\}$$

, and others.

- `template<typename _Tp >`
`_Tp std::__detail::__log_pochhammer_u (_Tp __a, _Tp __n)`

Return the logarithm of the (upper) Pochhammer symbol or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(n)$$

Thus this function returns

$$\ln[(a)_n] = \Gamma(a+n) - \Gamma(n), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

- `template<typename _Tp >`
`_Tp std::__detail::__pochhammer_l (_Tp __a, _Tp __n)`

Return the logarithm of the lower Pochhammer symbol or the falling factorial function. The lower Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular, $\ln(n)_n = n!$.

- `template<typename _Tp >`
`_Tp std::__detail::__pochhammer_u (_Tp __a, _Tp __n)`

Return the (upper) Pochhammer function or the rising factorial function. The Pochhammer symbol is defined by

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(n)$$

Many notations exist:

$$a^{\overline{n}}$$

,

$$\left[\begin{matrix} a \\ n \end{matrix} \right]$$

, and others.

- `template<typename _Tp >`
`_Tp std::__detail::__psi (_Tp __x)`

Return the digamma function. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

For negative argument the reflection formula is used:

$$\psi(x) = \psi(1-x) - \pi \cot(\pi x)$$

.

- `template<typename _Tp >`
`_Tp std::__detail::__psi (unsigned int __n, _Tp __x)`

Return the polygamma function $\psi^{(n)}(x)$.

- `template<typename _Tp >`
`_Tp std::__detail::__psi_asymp (_Tp __x)`

Return the digamma function for large argument. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

- `template<typename _Tp >`
`_Tp std::__detail::__psi_series (_Tp __x)`

Return the digamma function by series expansion. The digamma or $\psi(x)$ function is defined by

$$\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$$

Variables

- `constexpr _Factorial_table< long double > std::__detail::__S_double_factorial_table [301]`
- `constexpr _Factorial_table< long double > std::__detail::__S_factorial_table [171]`
- `constexpr _Factorial_table< long double > std::__detail::__S_neg_double_factorial_table [999]`
- `template<typename _Tp >`
`constexpr std::size_t std::__detail::__S_num_double_factorials = 0`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_double_factorials< double > = 301`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_double_factorials< float > = 57`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_double_factorials< long double > = 301`
- `template<typename _Tp >`
`constexpr std::size_t std::__detail::__S_num_factorials = 0`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_factorials< double > = 171`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_factorials< float > = 35`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_factorials< long double > = 171`
- `template<typename _Tp >`
`constexpr std::size_t std::__detail::__S_num_neg_double_factorials = 0`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_neg_double_factorials< double > = 150`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_neg_double_factorials< float > = 27`
- `template<>`
`constexpr std::size_t std::__detail::__S_num_neg_double_factorials< long double > = 999`

10.10.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.10.2 Macro Definition Documentation

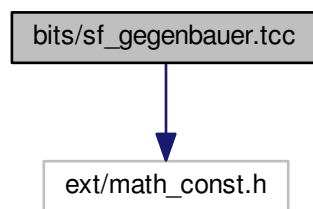
10.10.2.1 `#define _GLIBCXX_BITS_SF_GAMMA_TCC 1`

Definition at line 49 of file `sf_gamma.tcc`.

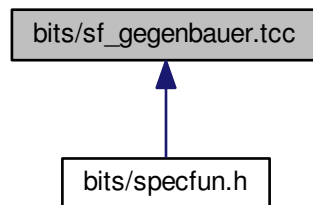
10.11 `bits/sf_gegenbauer.tcc` File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_gegenbauer.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_SF_GEGENBAUER_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__gegenbauer_poly` (unsigned int __n, _Tp __alpha, _Tp __x)

10.11.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

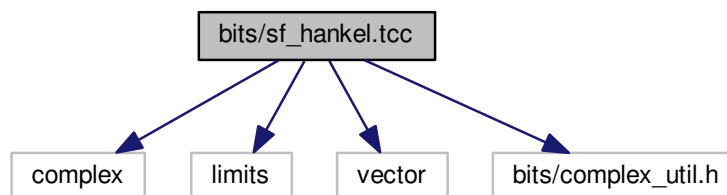
10.11.2 Macro Definition Documentation

10.11.2.1 `#define _GLIBCXX_SF_GEGENBAUER_TCC 1`

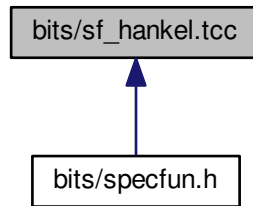
Definition at line 31 of file `sf_gegenbauer.tcc`.

10.12 bits/sf_hankel.tcc File Reference

```
#include <complex>
#include <limits>
#include <vector>
#include <bits/complex_util.h>
Include dependency graph for sf_hankel.tcc:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- `#define _GLIBCXX_BITS_SF_HANKEL_TCC 1`

Functions

- `template<typename _Tp >`
`void std::__detail::__airy_arg (std::complex< _Tp > __num2d3, std::complex< _Tp > __zeta, std::complex< _Tp > &__argp, std::complex< _Tp > &__argm)`
Compute the arguments for the Airy function evaluations carefully to prevent premature overflow. Note that the major work here is in `safe_div`. A faster, but less safe implementation can be obtained without use of `safe_div`.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_bessel (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Bessel function.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_1 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Hankel function of the first kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_hankel_2 (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Hankel function of the second kind.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__cyl_neumann (std::complex< _Tp > __nu, std::complex< _Tp > __z)`
Return the complex cylindrical Neumann function.
- `template<typename _Tp >`
`void std::__detail::__debye_region (std::complex< _Tp > __alpha, int &__indexr, char &__aorb)`
- `template<typename _Tp >`
`void std::__detail::__hankel (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`

- `template<typename _Tp >`
`void std::__detail::__hankel_debye (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > __alpha, int __indexr, char &__aorb, int &__morn, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
- `template<typename _Tp >`
`void std::__detail::__hankel_params (std::complex< _Tp > __nu, std::complex< _Tp > __zhat, std::complex< _Tp > &_p, std::complex< _Tp > &_p2, std::complex< _Tp > &_nup2, std::complex< _Tp > &_num2, std::complex< _Tp > &_num1d3, std::complex< _Tp > &_num2d3, std::complex< _Tp > &_num4d3, std::complex< _Tp > &_zeta, std::complex< _Tp > &_zetaphf, std::complex< _Tp > &_zetamhf, std::complex< _Tp > &_zetam3hf, std::complex< _Tp > &_zetrat)`
Compute parameters depending on z and nu that appear in the uniform asymptotic expansions of the Hankel functions and their derivatives, except the arguments to the Airy functions.
- `template<typename _Tp >`
`void std::__detail::__hankel_uniform (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
This routine computes the uniform asymptotic approximations of the Hankel functions and their derivatives including a patch for the case when the order equals or nearly equals the argument. At such points, Olver's expressions have zero denominators (and numerators) resulting in numerical problems. This routine averages results from four surrounding points in the complex plane to obtain the result in such cases.
- `template<typename _Tp >`
`void std::__detail::__hankel_uniform_olver (std::complex< _Tp > __nu, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H2, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2p)`
Compute approximate values for the Hankel functions of the first and second kinds using Olver's uniform asymptotic expansion to of order nu along with their derivatives.
- `template<typename _Tp >`
`void std::__detail::__hankel_uniform_outer (std::complex< _Tp > __nu, std::complex< _Tp > __z, _Tp __eps, std::complex< _Tp > &_zhat, std::complex< _Tp > &_1dnsq, std::complex< _Tp > &_num1d3, std::complex< _Tp > &_num2d3, std::complex< _Tp > &_p, std::complex< _Tp > &_p2, std::complex< _Tp > &_etm3h, std::complex< _Tp > &_etrat, std::complex< _Tp > &_Aip, std::complex< _Tp > &_o4dp, std::complex< _Tp > &_Aim, std::complex< _Tp > &_o4dm, std::complex< _Tp > &_od2p, std::complex< _Tp > &_od0dp, std::complex< _Tp > &_od2m, std::complex< _Tp > &_od0dm)`
Compute outer factors and associated functions of z and nu appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of z and nu returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.
- `template<typename _Tp >`
`void std::__detail::__hankel_uniform_sum (std::complex< _Tp > __p, std::complex< _Tp > __p2, std::complex< _Tp > __num2, std::complex< _Tp > __zetam3hf, std::complex< _Tp > _Aip, std::complex< _Tp > _o4dp, std::complex< _Tp > _Aim, std::complex< _Tp > _o4dm, std::complex< _Tp > _od2p, std::complex< _Tp > _od0dp, std::complex< _Tp > _od2m, std::complex< _Tp > _od0dm, _Tp __eps, std::complex< _Tp > &_H1sum, std::complex< _Tp > &_H1psum, std::complex< _Tp > &_H2sum, std::complex< _Tp > &_H2psum)`
Compute the sums in appropriate linear combinations appearing in Olver's uniform asymptotic expansions for the Hankel functions of the first and second kinds and their derivatives, using up to nterms (less than 5) to achieve relative error eps.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_bessel (unsigned int __n, std::complex< _Tp > __z)`
Return the complex spherical Bessel function.
- `template<typename _Tp >`
`void std::__detail::__sph_hankel (unsigned int __n, std::complex< _Tp > __z, std::complex< _Tp > &_H1, std::complex< _Tp > &_H1p, std::complex< _Tp > &_H2, std::complex< _Tp > &_H2p)`
Helper to compute complex spherical Hankel functions and their derivatives.
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_1 (unsigned int __n, std::complex< _Tp > __z)`

Return the complex spherical Hankel function of the first kind.

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_hankel_2 (unsigned int __n, std::complex< _Tp > __z)`

Return the complex spherical Hankel function of the second kind.

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_neumann (unsigned int __n, std::complex< _Tp > __z)`

Return the complex spherical Neumann function.

10.12.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

10.12.2 Macro Definition Documentation

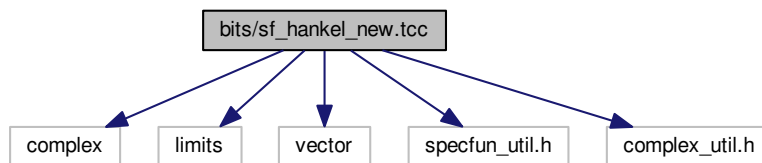
10.12.2.1 `#define _GLIBCXX_BITS_SF_HANKEL_TCC 1`

Definition at line 31 of file `sf_hankel.tcc`.

10.13 `bits/sf_hankel_new.tcc` File Reference

```
#include <complex>
#include <limits>
#include <vector>
#include "specfun_util.h"
#include "complex_util.h"
```

Include dependency graph for `sf_hankel_new.tcc`:



Macros

- `#define _GLIBCXX_BITS_SF_HANKEL_NEW_TCC 1`

10.13.1 Macro Definition Documentation

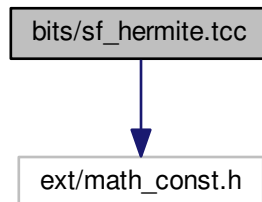
10.13.1.1 `#define _GLIBCXX_BITS_SF_HANKEL_NEW_TCC 1`

Definition at line 31 of file `sf_hankel_new.tcc`.

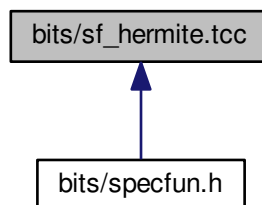
10.14 bits/sf_hermite.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_hermite.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_HERMITE_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__poly_hermite \(unsigned int __n, _Tp __x\)](#)
This routine returns the Hermite polynomial of order n : $H_n(x)$.

- `template<typename _Tp >`
`_Tp std::__detail::__poly_hermite_asymp` (unsigned int __n, _Tp __x)
This routine returns the Hermite polynomial of large order n : $H_n(x)$. We assume here that $x \geq 0$.
- `template<typename _Tp >`
`_Tp std::__detail::__poly_hermite_recursion` (unsigned int __n, _Tp __x)
This routine returns the Hermite polynomial of order n : $H_n(x)$ by recursion on n .

10.14.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

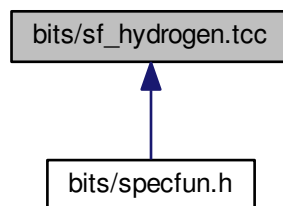
10.14.2 Macro Definition Documentation

10.14.2.1 `#define _GLIBCXX_BITS_SF_HERMITE_TCC 1`

Definition at line 42 of file `sf_hermite.tcc`.

10.15 `bits/sf_hydrogen.tcc` File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_BITS_SF_HYDROGEN_TCC 1`

Functions

- `template<typename _Tp >
std::complex< _Tp > std::__detail::__hydrogen (const unsigned int __n, const unsigned int __l, const unsigned int __m, const _Tp __Z, const _Tp __r, const _Tp __theta, const _Tp __phi)`

10.15.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

10.15.2 Macro Definition Documentation

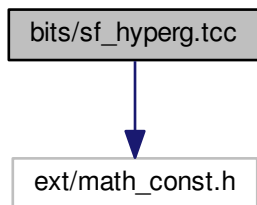
10.15.2.1 `#define _GLIBCXX_BITS_SF_HYDROGEN_TCC 1`

Definition at line 31 of file `sf_hydrogen.tcc`.

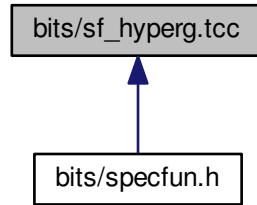
10.16 bits/sf_hyperg.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_hyperg.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- `#define GLIBCXX_BITS_SF_HYPERG_TCC 1`

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__conf_hyperg (_Tp __a, _Tp __c, _Tp __x)`
Return the confluent hypergeometric function ${}_1F_1(a; c; x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__conf_hyperg_lim (_Tp __c, _Tp __x)`
Return the confluent hypergeometric limit function ${}_0F_1(-; c; x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__conf_hyperg_lim_series (_Tp __c, _Tp __x)`
This routine returns the confluent hypergeometric limit function by series expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__conf_hyperg_luke (_Tp __a, _Tp __c, _Tp __xin)`
Return the hypergeometric function ${}_1F_1(a; c; x)$ by an iterative procedure described in Luke, Algorithms for the Computation of Mathematical Functions.
- `template<typename _Tp >`
`_Tp std::__detail::__conf_hyperg_series (_Tp __a, _Tp __c, _Tp __x)`
This routine returns the confluent hypergeometric function by series expansion.
- `template<typename _Tp >`
`_Tp std::__detail::__hyperg (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`
Return the hypergeometric function ${}_2F_1(a, b; c; x)$.
- `template<typename _Tp >`
`_Tp std::__detail::__hyperg_luke (_Tp __a, _Tp __b, _Tp __c, _Tp __xin)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by an iterative procedure described in Luke, *Algorithms for the Computation of Mathematical Functions*.

- `template<typename _Tp >`
`_Tp std::__detail::__hyperg_reflect (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by the reflection formulae in Abramowitz & Stegun formula 15.3.6 for $d = c - a - b$ not integral and formula 15.3.11 for $d = c - a - b$ integral. This assumes $a, b, c \neq$ negative integer.

- `template<typename _Tp >`
`_Tp std::__detail::__hyperg_series (_Tp __a, _Tp __b, _Tp __c, _Tp __x)`

Return the hypergeometric function ${}_2F_1(a, b; c; x)$ by series expansion.

10.16.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.16.2 Macro Definition Documentation

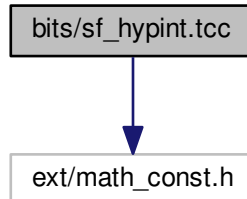
10.16.2.1 `#define _GLIBCXX_BITS_SF_HYPERG_TCC 1`

Definition at line 44 of file `sf_hyperg.tcc`.

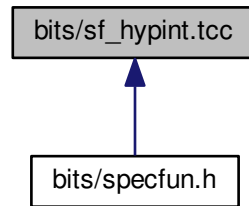
10.17 bits/sf_hypint.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_hypint.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_SF_HYPINT_TCC 1](#)

Functions

- `template<typename _Tp >`
`std::pair< _Tp, _Tp > std::__detail::__chshint (_Tp __x, _Tp &_Chi, _Tp &_Shi)`
This function returns the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals as a pair.
- `template<typename _Tp >`
`void std::__detail::__chshint_cont_frac (_Tp __t, _Tp &_Chi, _Tp &_Shi)`
This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by continued fraction for positive argument.
- `template<typename _Tp >`
`void std::__detail::__chshint_series (_Tp __t, _Tp &_Chi, _Tp &_Shi)`
This function computes the hyperbolic cosine $Chi(x)$ and hyperbolic sine $Shi(x)$ integrals by series summation for positive argument.

10.17.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.17.2 Macro Definition Documentation

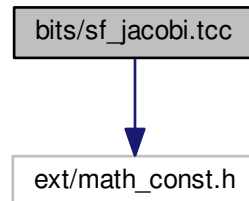
10.17.2.1 [#define _GLIBCXX_SF_HYPINT_TCC 1](#)

Definition at line 31 of file `sf_hypint.tcc`.

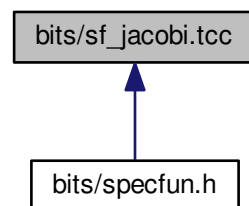
10.18 bits/sf_jacobi.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_jacobi.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_SF_JACOBI_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__poly_jacobi](#) (unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)

- `template<typename _Tp >`
`_Tp std::__detail::__poly_radial_jacobi` (unsigned int __n, unsigned int __m, _Tp __rho)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > std::__detail::__zernike` (unsigned int __n, int __m, _Tp __rho, _Tp __phi)

10.18.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.18.2 Macro Definition Documentation

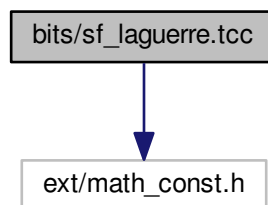
10.18.2.1 `#define _GLIBCXX_SF_JACOBI_TCC 1`

Definition at line 31 of file `sf_jacobi.tcc`.

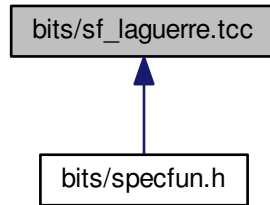
10.19 bits/sf_laguerre.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_laguerre.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__assoc_laguerre](#) (unsigned int __n, unsigned int __m, _Tp __x)
This routine returns the associated Laguerre polynomial of order n , degree m : $L_n^m(x)$.
- [template<typename _Tp > _Tp std::__detail::__laguerre](#) (unsigned int __n, _Tp __x)
This routine returns the Laguerre polynomial of order n : $L_n(x)$.
- [template<typename _Tpa, typename _Tp > _Tp std::__detail::__poly_laguerre](#) (unsigned int __n, _Tpa __alpha1, _Tp __x)
This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^\alpha(x)$.
- [template<typename _Tpa, typename _Tp > _Tp std::__detail::__poly_laguerre_hyperg](#) (unsigned int __n, _Tpa __alpha1, _Tp __x)
Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.
- [template<typename _Tpa, typename _Tp > _Tp std::__detail::__poly_laguerre_large_n](#) (unsigned __n, _Tpa __alpha1, _Tp __x)
This routine returns the associated Laguerre polynomial of order n , degree $\alpha > -1$ for large n . Abramowitz & Stegun, 13.5.21.
- [template<typename _Tpa, typename _Tp > _Tp std::__detail::__poly_laguerre_recursion](#) (unsigned int __n, _Tpa __alpha1, _Tp __x)
This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^\alpha(x)$ by recursion.

10.19.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.19.2 Macro Definition Documentation

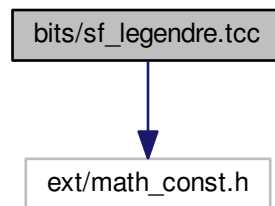
10.19.2.1 `#define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1`

Definition at line 44 of file `sf_laguerre.tcc`.

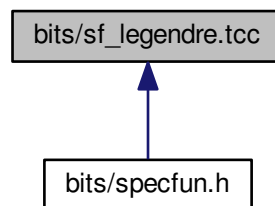
10.20 `bits/sf_legendre.tcc` File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_legendre.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)

- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1](#)

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__assoc_legendre_p (unsigned int __l, unsigned int __m, _Tp __x)`
Return the associated Legendre function by recursion on l and downward recursion on m .
- `template<typename _Tp >`
`_Tp std::__detail::__legendre_q (unsigned int __l, _Tp __x)`
Return the Legendre function of the second kind by upward recursion on order l .
- `template<typename _Tp >`
`_Tp std::__detail::__poly_legendre_p (unsigned int __l, _Tp __x)`
Return the Legendre polynomial by upward recursion on order l .
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__sph_harmonic (unsigned int __l, int __m, _Tp __theta, _Tp __phi)`
Return the spherical harmonic function.
- `template<typename _Tp >`
`_Tp std::__detail::__sph_legendre (unsigned int __l, unsigned int __m, _Tp __theta)`
Return the spherical associated Legendre function.

10.20.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.20.2 Macro Definition Documentation

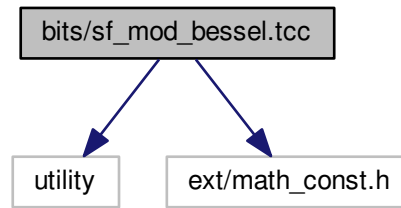
10.20.2.1 [#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1](#)

Definition at line 47 of file `sf_legendre.tcc`.

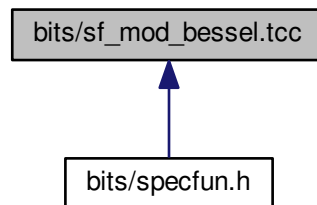
10.21 bits/sf_mod_bessel.tcc File Reference

```
#include <utility>
#include <ext/math_const.h>
```

Include dependency graph for `sf_mod_bessel.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1`

Functions

- `template<typename _Tp >`
`void std::__detail::__airy (_Tp __z, _Tp &_Ai, _Tp &_Bi, _Tp &_Aip, _Tp &_Bip)`
Compute the Airy functions $Ai(x)$ and $Bi(x)$ and their first derivatives $Ai'(x)$ and $Bi'(x)$ respectively.
- `template<typename _Tp >`
`_Tp std::__detail::__cyl_bessel_i (_Tp __nu, _Tp __x)`
Return the regular modified Bessel function of order ν : $I_\nu(x)$.

- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_ik (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`
Return the modified cylindrical Bessel functions and their derivatives of order ν by various means.
- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_ik_asymp (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`
This routine computes the asymptotic modified cylindrical Bessel and functions of order ν : $I_\nu(x)$, $N_\nu(x)$. Use this for $x \gg \nu^2 + 1$.
- `template<typename _Tp >`
`void std::__detail::__cyl_bessel_ik_steeds (_Tp __nu, _Tp __x, _Tp &_Inu, _Tp &_Knu, _Tp &_Ipnu, _Tp &_Kpnu)`
Compute the modified Bessel functions $I_\nu(x)$ and $K_\nu(x)$ and their first derivatives $I'_\nu(x)$ and $K'_\nu(x)$ respectively. These four functions are computed together for numerical stability.
- `template<typename _Tp >`
`_Tp std::__detail::__cyl_bessel_k (_Tp __nu, _Tp __x)`
Return the irregular modified Bessel function $K_\nu(x)$ of order ν .
- `template<typename _Tp >`
`void std::__detail::__fock_airy (_Tp __x, std::complex< _Tp > &__w1, std::complex< _Tp > &__w2, std::complex< _Tp > &__w1p, std::complex< _Tp > &__w2p)`
Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w'_1(x)$ and $w'_2(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$
- `template<typename _Tp >`
`void std::__detail::__sph_bessel_ik (unsigned int __n, _Tp __x, _Tp &__i_n, _Tp &__k_n, _Tp &__ip_n, _Tp &__kp_n)`
Compute the spherical modified Bessel functions $i_n(x)$ and $k_n(x)$ and their first derivatives $i'_n(x)$ and $k'_n(x)$ respectively.

10.21.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

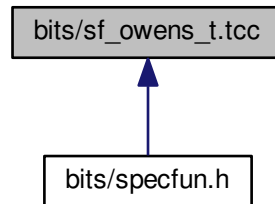
10.21.2 Macro Definition Documentation

10.21.2.1 `#define _GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1`

Definition at line 47 of file `sf_mod_bessel.tcc`.

10.22 bits/sf_owens_t.tcc File Reference

This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_OWENS_T_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__gauss \(_Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__owens_t \(_Tp __h, _Tp __a\)](#)
- [template<typename _Tp > _Tp std::__detail::__znorm1 \(_Tp __x\)](#)
- [template<typename _Tp > _Tp std::__detail::__znorm2 \(_Tp __x\)](#)

10.22.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

10.22.2 Macro Definition Documentation

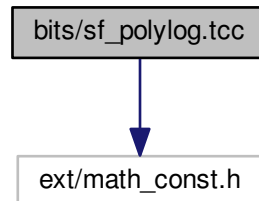
10.22.2.1 [#define _GLIBCXX_BITS_SF_OWENS_T_TCC 1](#)

Definition at line 31 of file `sf_owens_t.tcc`.

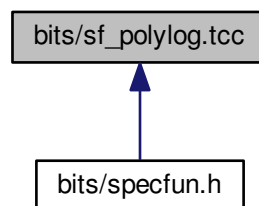
10.23 bits/sf_polylog.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for sf_polylog.tcc:



This graph shows which files directly or indirectly include this file:



Namespaces

- `std`
- `std::__detail`

Macros

- `#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1`

Functions

- `template<typename _Tp >`
`_Tp std::__detail::__bose_einstein (_Tp __s, _Tp __x)`

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__clamp_0_m2pi (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__clamp_pi (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__clausen (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__clausen (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__clausen_c (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__clausen_c (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__clausen_s (unsigned int __m, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__clausen_s (unsigned int __m, _Tp __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__dirichlet_beta (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__dirichlet_beta (_Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__dirichlet_eta (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__dirichlet_eta (_Tp __w)`
- `template<typename _Tp >`
`_Tp std::__detail::__fermi_dirac (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`bool std::__detail::__fpequal (const _Tp &__a, const _Tp &__b)`
- `template<typename _Tp >`
`bool std::__detail::__fpimag (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`
`bool std::__detail::__fpimag (const _Tp)`
- `template<typename _Tp >`
`bool std::__detail::__fpreal (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`
`bool std::__detail::__fpreal (const _Tp)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__hurwitz_zeta (_Tp __s, std::complex< _Tp > __a)`
- `template<typename _Tp >`
`_Tp std::__detail::__polylog (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp, typename ArgType >`
`__gnu_cxx::__promote_num_t< std::complex< _Tp >, ArgType > std::__detail::__polylog_exp (_Tp __s, ArgType __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_asymp (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_int_neg (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_int_neg (const int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_int_pos (unsigned int __s, std::complex< _Tp > __w)`

- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_int_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_neg (int __s, std::complex< _Tp > __w)`
- `template<typename _Tp, int __sigma>`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_even (unsigned int __n, std::complex< _Tp > __w)`
- `template<typename _Tp, int __sigma>`
`std::complex< _Tp > std::__detail::__polylog_exp_neg_odd (unsigned int __n, std::complex< _Tp > __w)`
- `template<typename _PowTp, typename _Tp >`
`_Tp std::__detail::__polylog_exp_negative_real_part (_PowTp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (unsigned int __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_real_neg (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_real_neg (_Tp __s, _Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_real_pos (_Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_real_pos (_Tp __s, _Tp __w)`
- `template<typename _Tp = double>`
`_Tp std::__detail::__evenzeta (unsigned int __k)`

10.23.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.23.2 Macro Definition Documentation

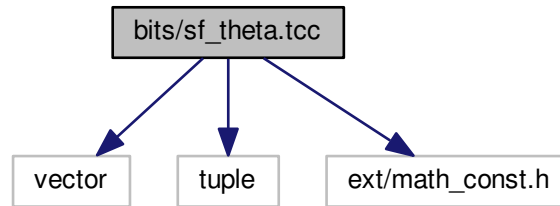
10.23.2.1 `#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1`

Definition at line 41 of file `sf_polylog.tcc`.

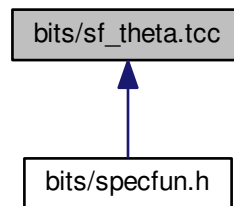
10.24 bits/sf_theta.tcc File Reference

```
#include <vector>
#include <tuple>
#include <ext/math_const.h>
```

Include dependency graph for `sf_theta.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_SF_THETA_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__ellnome \(_Tp __k\)](#)
- [template<typename _Tp > _Tp std::__detail::__ellnome_k \(_Tp __k\)](#)
- [template<typename _Tp > _Tp std::__detail::__ellnome_series \(_Tp __k\)](#)

- `template<typename _Tp >`
`std::tuple< _Tp, _Tp, _Tp > std::__detail::__jacobi_sncndn (_Tp __k, _Tp __u)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_1 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_2 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_2_asymp (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_2_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_3 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_3_asymp (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_3_sum (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_4 (_Tp __nu, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_c (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_d (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_n (_Tp __k, _Tp __x)`
- `template<typename _Tp >`
`_Tp std::__detail::__theta_s (_Tp __k, _Tp __x)`

10.24.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

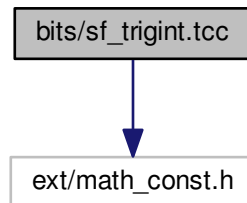
10.24.2 Macro Definition Documentation

10.24.2.1 `#define _GLIBCXX_SF_THETA_TCC 1`

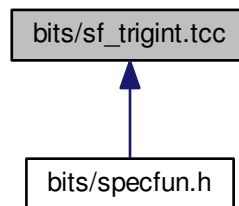
Definition at line 31 of file `sf_theta.tcc`.

10.25 bits/sf_trigint.tcc File Reference

```
#include <ext/math_const.h>  
Include dependency graph for sf_trigint.tcc:
```



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_SF_TRIGINT_TCC 1](#)

Enumerations

- [enum { std::__detail::SININT, std::__detail::COSINT }](#)

Functions

- `template<typename _Tp >
std::pair< _Tp, _Tp > std::__detail::__sincosint (_Tp __x)`
This function returns the sine $Si(x)$ and cosine $Ci(x)$ integrals as a pair.
- `template<typename _Tp >
void std::__detail::__sincosint_asymp (_Tp __t, _Tp &_Si, _Tp &_Ci)`
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.
- `template<typename _Tp >
void std::__detail::__sincosint_cont_frac (_Tp __t, _Tp &_Si, _Tp &_Ci)`
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by continued fraction for positive argument.
- `template<typename _Tp >
void std::__detail::__sincosint_series (_Tp __t, _Tp &_Si, _Tp &_Ci)`
This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by series summation for positive argument.

10.25.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.25.2 Macro Definition Documentation

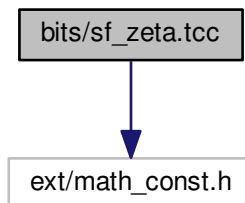
10.25.2.1 `#define _GLIBCXX_SF_TRIGINT_TCC 1`

Definition at line 31 of file `sf_trigint.tcc`.

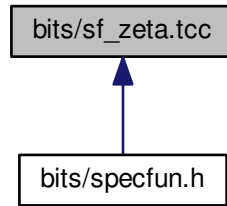
10.26 bits/sf_zeta.tcc File Reference

```
#include <ext/math_const.h>
```

Include dependency graph for `sf_zeta.tcc`:



This graph shows which files directly or indirectly include this file:



Namespaces

- [std](#)
- [std::__detail](#)

Macros

- [#define _GLIBCXX_BITS_SF_ZETA_TCC 1](#)

Functions

- [template<typename _Tp > _Tp std::__detail::__dilog \(_Tp __x\)](#)
Compute the dilogarithm function $Li_2(x)$ by summation for $x \leq 1$.
- [template<typename _Tp > _Tp std::__detail::__hurwitz_zeta \(_Tp __s, _Tp __a\)](#)
Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.
- [template<typename _Tp > _Tp std::__detail::__hurwitz_zeta_euler_maclaurin \(_Tp __s, _Tp __a\)](#)
Return the Hurwitz zeta function $\zeta(s, a)$ for all $s \neq 1$ and $a > -1$.
- [template<typename _Tp > _Tp std::__detail::__riemann_zeta \(_Tp __s\)](#)
Return the Riemann zeta function $\zeta(s)$.
- [template<typename _Tp > _Tp std::__detail::__riemann_zeta_alt \(_Tp __s\)](#)
Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.
- [template<typename _Tp > _Tp std::__detail::__riemann_zeta_euler_maclaurin \(_Tp __s\)](#)
Evaluate the Riemann zeta function $\zeta(s)$ by an alternate series for $s > 0$.
- [template<typename _Tp > _Tp std::__detail::__riemann_zeta_glob \(_Tp __s\)](#)
Evaluate the Riemann zeta function by series for all $s \neq 1$. Convergence is great until largish negative numbers. Then the convergence of the > 0 sum gets better.

- `template<typename _Tp >`
`_Tp std::__detail::__riemann_zeta_m_1 (_Tp __s)`
Return the Riemann zeta function $\zeta(s) - 1$.
- `template<typename _Tp >`
`_Tp std::__detail::__riemann_zeta_m_1_sum (_Tp __s)`
Return the Riemann zeta function $\zeta(s) - 1$ by summation for $s > 1$. This is a small remainder for large s .
- `template<typename _Tp >`
`_Tp std::__detail::__riemann_zeta_product (_Tp __s)`
Compute the Riemann zeta function $\zeta(s)$ using the product over prime factors.
- `template<typename _Tp >`
`_Tp std::__detail::__riemann_zeta_sum (_Tp __s)`
Compute the Riemann zeta function $\zeta(s)$ by summation for $s > 1$.

Variables

- `constexpr size_t std::__detail::__Num_Euler_Maclaurin_zeta = 100`
- `constexpr long double std::__detail::__S_Euler_Maclaurin_zeta [_Num_Euler_Maclaurin_zeta]`
- `constexpr size_t std::__detail::__S_num_zetam1 = 33`
- `constexpr long double std::__detail::__S_zetam1 [_S_num_zetam1]`

10.26.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.26.2 Macro Definition Documentation

10.26.2.1 `#define _GLIBCXX_BITS_SF_ZETA_TCC 1`

Definition at line 46 of file `sf_zeta.tcc`.

10.27 bits/specfun.h File Reference

```
#include <bits/c++config.h>
#include <limits>
#include <bits/stl_algobase.h>
#include <bits/specfun_util.h>
#include <type_traits>
#include <bits/numeric_limits.h>
#include <bits/complex_util.h>
#include <bits/sf_gamma.tcc>
#include <bits/sf_bessel.tcc>
#include <bits/sf_beta.tcc>
#include <bits/sf_cardinal.tcc>
#include <bits/sf_chebyshev.tcc>
#include <bits/sf_dawson.tcc>
#include <bits/sf_ellint.tcc>
#include <bits/sf_expint.tcc>
#include <bits/sf_fresnel.tcc>
#include <bits/sf_gegenbauer.tcc>
#include <bits/sf_hyperg.tcc>
#include <bits/sf_hypint.tcc>
#include <bits/sf_jacobi.tcc>
#include <bits/sf_laguerre.tcc>
#include <bits/sf_legendre.tcc>
#include <bits/sf_hydrogen.tcc>
#include <bits/sf_mod_bessel.tcc>
#include <bits/sf_hermite.tcc>
#include <bits/sf_theta.tcc>
#include <bits/sf_trigint.tcc>
#include <bits/sf_zeta.tcc>
#include <bits/sf_owens_t.tcc>
#include <bits/sf_polylog.tcc>
#include <bits/sf_airy.tcc>
#include <bits/sf_hankel.tcc>
```

Include dependency graph for specfun.h:



Namespaces

- [__gnu_cxx](#)
- [std](#)

Macros

- [#define __cpp_lib_math_special_functions 201603L](#)
- [#define __STDCPP_MATH_SPEC_FUNCS__ 201003L](#)

Enumerations

- enum { [__gnu_cxx::GLIBCXX_JACOBI_SN](#), [__gnu_cxx::GLIBCXX_JACOBI_CN](#), [__gnu_cxx::GLIBCXX_JACOBI_DN](#) }

Functions

- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::airy_ai](#) (_Tp __x)
- float [__gnu_cxx::airy_aif](#) (float __x)
- long double [__gnu_cxx::airy_ail](#) (long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::airy_bi](#) (_Tp __x)
- float [__gnu_cxx::airy_bif](#) (float __x)
- long double [__gnu_cxx::airy_bil](#) (long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote<_Tp>::__type](#) [std::assoc_laguerre](#) (unsigned int __n, unsigned int __m, _Tp __x)
- float [std::assoc_laguerref](#) (unsigned int __n, unsigned int __m, float __x)
- long double [std::assoc_laguerrel](#) (unsigned int __n, unsigned int __m, long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote<_Tp>::__type](#) [std::assoc_legendre](#) (unsigned int __l, unsigned int __m, _Tp __x)
- float [std::assoc_legendref](#) (unsigned int __l, unsigned int __m, float __x)
- long double [std::assoc_legendrel](#) (unsigned int __l, unsigned int __m, long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::bernoulli](#) (unsigned int __n)
- float [__gnu_cxx::bernoullif](#) (unsigned int __n)
- long double [__gnu_cxx::bernoullil](#) (unsigned int __n)
- template<typename _Tpa, typename _Tpb >
 [__gnu_cxx::__promote_2<_Tpa, _Tpb>::__type](#) [std::beta](#) (_Tpa __a, _Tpb __b)
- float [std::betaf](#) (float __a, float __b)
- long double [std::betal](#) (long double __a, long double __b)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::bincoef](#) (unsigned int __n, unsigned int __k)
- float [__gnu_cxx::bincoefficient](#) (unsigned int __n, unsigned int __k)
- long double [__gnu_cxx::bincoefficientl](#) (unsigned int __n, unsigned int __k)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::chebyshev_t](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_tf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_tl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::chebyshev_u](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_uf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_ul](#) (unsigned int __n, long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::chebyshev_v](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_vf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_vl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
 [__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::chebyshev_w](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_wf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_wl](#) (unsigned int __n, long double __x)

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen` (unsigned int __m, _Tp __w)
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::clausen` (unsigned int __m, std::complex< _Tp > __w)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_c` (unsigned int __m, _Tp __w)
- `float __gnu_cxx::clausen_cf` (unsigned int __m, float __w)
- `long double __gnu_cxx::clausen_cl` (unsigned int __m, long double __w)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_s` (unsigned int __m, _Tp __w)
- `float __gnu_cxx::clausen_sf` (unsigned int __m, float __w)
- `long double __gnu_cxx::clausen_sl` (unsigned int __m, long double __w)
- `float __gnu_cxx::clausenf` (unsigned int __m, float __w)
- `std::complex< float > __gnu_cxx::clausenf` (unsigned int __m, std::complex< float > __w)
- `long double __gnu_cxx::clausenl` (unsigned int __m, long double __w)
- `std::complex< long double > __gnu_cxx::clausenl` (unsigned int __m, std::complex< long double > __w)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_1` (_Tp __k)
- `float std::comp_ellint_1f` (float __k)
- `long double std::comp_ellint_1l` (long double __k)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_2` (_Tp __k)
- `float std::comp_ellint_2f` (float __k)
- `long double std::comp_ellint_2l` (long double __k)
- `template<typename _Tp, typename _Tpn >`
`__gnu_cxx::__promote_2< _Tp, _Tpn >::__type std::comp_ellint_3` (_Tp __k, _Tpn __nu)
- `float std::comp_ellint_3f` (float __k, float __nu)
- Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for float modulus k.*
- `long double std::comp_ellint_3l` (long double __k, long double __nu)
- Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for long double modulus k.*
- `template<typename _Tk >`
`__gnu_cxx::__promote_num_t< _Tk > __gnu_cxx::comp_ellint_d` (_Tk __k)
- `float __gnu_cxx::comp_ellint_df` (float __k)
- `long double __gnu_cxx::comp_ellint_dl` (long double __k)
- `float __gnu_cxx::comp_ellint_rf` (float __x, float __y)
- `long double __gnu_cxx::comp_ellint_rf` (long double __x, long double __y)
- `template<typename _Tx, typename _Ty >`
`__gnu_cxx::__promote_num_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rf` (_Tx __x, _Ty __y)
- `float __gnu_cxx::comp_ellint_rg` (float __x, float __y)
- `long double __gnu_cxx::comp_ellint_rg` (long double __x, long double __y)
- `template<typename _Tx, typename _Ty >`
`__gnu_cxx::__promote_num_t< _Tx, _Ty > __gnu_cxx::comp_ellint_rg` (_Tx __x, _Ty __y)
- `template<typename _Tpa, typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_3< _Tpa, _Tpc, _Tp >::__type __gnu_cxx::conf_hyperg` (_Tpa __a, _Tpc __c, _Tp __x)
- `template<typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpc, _Tp >::__type __gnu_cxx::conf_hyperg_lim` (_Tpc __c, _Tp __x)
- `float __gnu_cxx::conf_hyperg_limf` (float __c, float __x)
- `long double __gnu_cxx::conf_hyperg_liml` (long double __c, long double __x)
- `float __gnu_cxx::conf_hypergf` (float __a, float __c, float __x)
- `long double __gnu_cxx::conf_hypergl` (long double __a, long double __c, long double __x)

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::coshint (_Tp __x)`
- `float __gnu_cxx::coshintf (float __x)`
- `long double __gnu_cxx::coshintl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::cosint (_Tp __x)`
- `float __gnu_cxx::cosintf (float __x)`
- `long double __gnu_cxx::cosintl (long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::type std::cyl_bessel_i (_Tpnu __nu, _Tp __x)`
- `float std::cyl_bessel_if (float __nu, float __x)`
- `long double std::cyl_bessel_il (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::type std::cyl_bessel_j (_Tpnu __nu, _Tp __x)`
- `float std::cyl_bessel_jf (float __nu, float __x)`
- `long double std::cyl_bessel_jl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::type std::cyl_bessel_k (_Tpnu __nu, _Tp __x)`
- `float std::cyl_bessel_kf (float __nu, float __x)`
- `long double std::cyl_bessel_kl (long double __nu, long double __x)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::type std::cyl_neumann (_Tpnu __nu, _Tp __x)`
- `float std::cyl_neumannf (float __nu, float __x)`
- `long double std::cyl_neumannl (long double __nu, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::dawson (_Tp __x)`
- `float __gnu_cxx::dawsonf (float __x)`
- `long double __gnu_cxx::dawsonl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::digamma (_Tp __z)`

- float `__gnu_cxx::digammaf` (float `__z`)
- long double `__gnu_cxx::digammal` (long double `__z`)
- template<typename `_Tp` >
`__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::dilog` (`_Tp` `__x`)
- float `__gnu_cxx::dilogf` (float `__x`)
- long double `__gnu_cxx::dilogl` (long double `__x`)
- template<typename `_Tp` >
`_Tp __gnu_cxx::dirichlet_beta` (`_Tp` `__s`)
- float `__gnu_cxx::dirichlet_betaf` (float `__s`)
- long double `__gnu_cxx::dirichlet_betall` (long double `__s`)
- template<typename `_Tp` >
`_Tp __gnu_cxx::dirichlet_eta` (`_Tp` `__s`)
- float `__gnu_cxx::dirichlet_etaf` (float `__s`)
- long double `__gnu_cxx::dirichlet_etall` (long double `__s`)
- template<typename `_Tp` >
`__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::double_factorial` (int `__n`)
- float `__gnu_cxx::double_factorialf` (int `__n`)
- long double `__gnu_cxx::double_factoriall` (int `__n`)
- template<typename `_Tp`, typename `_Tpp` >
`__gnu_cxx::__promote_2<_Tp, _Tpp>::__type std::ellint_1` (`_Tp` `__k`, `_Tpp` `__phi`)
- float `std::ellint_1f` (float `__k`, float `__phi`)
- long double `std::ellint_1l` (long double `__k`, long double `__phi`)
- template<typename `_Tp`, typename `_Tpp` >
`__gnu_cxx::__promote_2<_Tp, _Tpp>::__type std::ellint_2` (`_Tp` `__k`, `_Tpp` `__phi`)
- float `std::ellint_2f` (float `__k`, float `__phi`)
- Return the incomplete elliptic integral of the second kind $E(k, \phi)$ for float argument.*
- long double `std::ellint_2l` (long double `__k`, long double `__phi`)
- Return the incomplete elliptic integral of the second kind $E(k, \phi)$.*
- template<typename `_Tp`, typename `_Tpn`, typename `_Tpp` >
`__gnu_cxx::__promote_3<_Tp, _Tpn, _Tpp>::__type std::ellint_3` (`_Tp` `__k`, `_Tpn` `__nu`, `_Tpp` `__phi`)
- Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.*
- float `std::ellint_3f` (float `__k`, float `__nu`, float `__phi`)
- Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ for float argument.*
- long double `std::ellint_3l` (long double `__k`, long double `__nu`, long double `__phi`)
- Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.*
- template<typename `_Tk`, typename `_Tp`, typename `_Ta`, typename `_Tb` >
`__gnu_cxx::__promote_num_t<_Tk, _Tp, _Ta, _Tb> __gnu_cxx::ellint_cel` (`_Tk` `__k_c`, `_Tp` `__p`, `_Ta` `__a`, `_Tb` `__b`)
- float `__gnu_cxx::ellint_celf` (float `__k_c`, float `__p`, float `__a`, float `__b`)
- long double `__gnu_cxx::ellint_cell` (long double `__k_c`, long double `__p`, long double `__a`, long double `__b`)
- template<typename `_Tk`, typename `_Tphi` >
`__gnu_cxx::__promote_num_t<_Tk, _Tphi> __gnu_cxx::ellint_d` (`_Tk` `__k`, `_Tphi` `__phi`)
- float `__gnu_cxx::ellint_df` (float `__k`, float `__phi`)
- long double `__gnu_cxx::ellint_dl` (long double `__k`, long double `__phi`)
- template<typename `_Tp`, typename `_Tk` >
`__gnu_cxx::__promote_num_t<_Tp, _Tk> __gnu_cxx::ellint_el1` (`_Tp` `__x`, `_Tk` `__k_c`)
- float `__gnu_cxx::ellint_el1f` (float `__x`, float `__k_c`)
- long double `__gnu_cxx::ellint_el1l` (long double `__x`, long double `__k_c`)
- template<typename `_Tp`, typename `_Tk`, typename `_Ta`, typename `_Tb` >
`__gnu_cxx::__promote_num_t<_Tp, _Tk, _Ta, _Tb> __gnu_cxx::ellint_el2` (`_Tp` `__x`, `_Tk` `__k_c`, `_Ta` `__a`, `_Tb` `__b`)

- float [__gnu_cxx::ellint_el2f](#) (float __x, float __k_c, float __a, float __b)
- long double [__gnu_cxx::ellint_el2l](#) (long double __x, long double __k_c, long double __a, long double __b)
- template<typename _Tx, typename _Tk, typename _Tp >
[__gnu_cxx::__promote_num_t<_Tx, _Tk, _Tp>](#) [__gnu_cxx::ellint_el3](#) (_Tx __x, _Tk __k_c, _Tp __p)
- float [__gnu_cxx::ellint_el3f](#) (float __x, float __k_c, float __p)
- long double [__gnu_cxx::ellint_el3l](#) (long double __x, long double __k_c, long double __p)
- template<typename _Tp, typename _Up >
[__gnu_cxx::__promote_num_t<_Tp, _Up>](#) [__gnu_cxx::ellint_rc](#) (_Tp __x, _Up __y)
- float [__gnu_cxx::ellint_rcf](#) (float __x, float __y)
- long double [__gnu_cxx::ellint_rcl](#) (long double __x, long double __y)
- template<typename _Tp, typename _Up, typename _Vp >
[__gnu_cxx::__promote_num_t<_Tp, _Up, _Vp>](#) [__gnu_cxx::ellint_rd](#) (_Tp __x, _Up __y, _Vp __z)
- float [__gnu_cxx::ellint_rdf](#) (float __x, float __y, float __z)
- long double [__gnu_cxx::ellint_rdl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
[__gnu_cxx::__promote_num_t<_Tp, _Up, _Vp>](#) [__gnu_cxx::ellint_rf](#) (_Tp __x, _Up __y, _Vp __z)
- float [__gnu_cxx::ellint_rff](#) (float __x, float __y, float __z)
- long double [__gnu_cxx::ellint_rfl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp >
[__gnu_cxx::__promote_num_t<_Tp, _Up, _Vp>](#) [__gnu_cxx::ellint_rg](#) (_Tp __x, _Up __y, _Vp __z)
- float [__gnu_cxx::ellint_rgf](#) (float __x, float __y, float __z)
- long double [__gnu_cxx::ellint_rgl](#) (long double __x, long double __y, long double __z)
- template<typename _Tp, typename _Up, typename _Vp, typename _Wp >
[__gnu_cxx::__promote_num_t<_Tp, _Up, _Vp, _Wp>](#) [__gnu_cxx::ellint_rj](#) (_Tp __x, _Up __y, _Vp __z, _Wp __p)
- float [__gnu_cxx::ellint_rjf](#) (float __x, float __y, float __z, float __p)
- long double [__gnu_cxx::ellint_rjl](#) (long double __x, long double __y, long double __z, long double __p)
- template<typename _Tp >
[_Tp](#) [__gnu_cxx::ellnome](#) (_Tp __k)
- float [__gnu_cxx::ellnomef](#) (float __k)
- long double [__gnu_cxx::ellnomel](#) (long double __k)
- template<typename _Tp >
[__gnu_cxx::__promote<_Tp>::type](#) [std::expint](#) (_Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::expint_e1](#) (_Tp __x)
- float [__gnu_cxx::expint_e1f](#) (float __x)
- long double [__gnu_cxx::expint_e1l](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::expint_en](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::expint_enf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::expint_enl](#) (unsigned int __n, long double __x)
- float [std::expintf](#) (float __x)
- long double [std::expintl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::factorial](#) (unsigned int __n)
- float [__gnu_cxx::factorialf](#) (unsigned int __n)
- long double [__gnu_cxx::factoriall](#) (unsigned int __n)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::fresnel_c](#) (_Tp __x)
- float [__gnu_cxx::fresnel_cf](#) (float __x)
- long double [__gnu_cxx::fresnel_cl](#) (long double __x)

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::fresnel_s (_Tp __x)`
- `float __gnu_cxx::fresnel_sf (float __x)`
- `long double __gnu_cxx::fresnel_sl (long double __x)`
- `template<typename _Tn , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tn, _Tp > __gnu_cxx::gamma_l (_Tn __n, _Tp __x)`
- `float __gnu_cxx::gamma_lf (float __n, float __x)`
- `long double __gnu_cxx::gamma_ll (long double __n, long double __x)`
- `template<typename _Ta , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Ta, _Tp > __gnu_cxx::gamma_p (_Ta __a, _Tp __x)`
- `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::__promote_num_t< _Ta, _Tp > __gnu_cxx::gamma_q (_Ta __a, _Tp __x)`
- `float __gnu_cxx::gamma_qf (float __a, float __x)`
- `long double __gnu_cxx::gamma_ql (long double __a, long double __x)`
- `template<typename _Tn , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tn, _Tp > __gnu_cxx::gamma_u (_Tn __n, _Tp __x)`
- `float __gnu_cxx::gamma_uf (float __n, float __x)`
- `long double __gnu_cxx::gamma_ul (long double __n, long double __x)`
- `template<typename _Talpha , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Talpha, _Tp > __gnu_cxx::gegenbauer (unsigned int __n, _Talpha __alpha, _Tp __x)`
- `float __gnu_cxx::gegenbauerf (unsigned int __n, float __alpha, float __x)`
- `long double __gnu_cxx::gegenbauerl (unsigned int __n, long double __alpha, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::hermite (unsigned int __n, _Tp __x)`
- `float std::hermitef (unsigned int __n, float __x)`
- `long double std::hermitel (unsigned int __n, long double __x)`
- `template<typename _Tk , typename _Tphi >`
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::heuman_lambda (_Tk __k, _Tphi __phi)`
- `float __gnu_cxx::heuman_lambdaf (float __k, float __phi)`
- `long double __gnu_cxx::heuman_lambdal (long double __k, long double __phi)`
- `template<typename _Tp , typename _Up >`
`__gnu_cxx::__promote_num_t< _Tp, _Up > __gnu_cxx::hurwitz_zeta (_Tp __s, _Up __a)`
- `template<typename _Tp , typename _Up >`
`std::complex< _Tp > __gnu_cxx::hurwitz_zeta (_Tp __s, std::complex< _Up > __a)`
- `float __gnu_cxx::hurwitz_zetaf (float __s, float __a)`
- `long double __gnu_cxx::hurwitz_zetal (long double __s, long double __a)`
- `template<typename _Tpa , typename _Tpb , typename _Tpc , typename _Tp >`
`__gnu_cxx::__promote_4< _Tpa, _Tpb, _Tpc, _Tp >::__type __gnu_cxx::hyperg (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)`
- `float __gnu_cxx::hypergf (float __a, float __b, float __c, float __x)`
- `long double __gnu_cxx::hypergl (long double __a, long double __b, long double __c, long double __x)`
- `template<typename _Ta , typename _Tb , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta (_Ta __a, _Tb __b, _Tp __x)`
- `template<typename _Ta , typename _Tb , typename _Tp >`
`__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibetac (_Ta __a, _Tb __b, _Tp __x)`
- `float __gnu_cxx::ibetaf (float __a, float __b, float __x)`
- `long double __gnu_cxx::ibetac (long double __a, long double __b, long double __x)`
- `float __gnu_cxx::ibetaf (float __a, float __b, float __x)`
- `long double __gnu_cxx::ibetal (long double __a, long double __b, long double __x)`

- `template<typename _Talpha, typename _Tbeta, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Talpha, _Tbeta, _Tp > __gnu_cxx::jacobi` (unsigned __n, _Talpha __alpha,
_Tbeta __beta, _Tp __x)
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_cn` (_Kp __k, _Up __u)
- `float __gnu_cxx::jacobi_cnf` (float __k, float __u)
- `long double __gnu_cxx::jacobi_cnl` (long double __k, long double __u)
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_dn` (_Kp __k, _Up __u)
- `float __gnu_cxx::jacobi_dnf` (float __k, float __u)
- `long double __gnu_cxx::jacobi_dnl` (long double __k, long double __u)
- `template<typename _Kp, typename _Up >`
`__gnu_cxx::__promote_num_t< _Kp, _Up > __gnu_cxx::jacobi_sn` (_Kp __k, _Up __u)
- `float __gnu_cxx::jacobi_snf` (float __k, float __u)
- `long double __gnu_cxx::jacobi_snl` (long double __k, long double __u)
- `template<typename _Tk, typename _Tphi >`
`__gnu_cxx::__promote_num_t< _Tk, _Tphi > __gnu_cxx::jacobi_zeta` (_Tk __k, _Tphi __phi)
- `float __gnu_cxx::jacobi_zetaf` (float __k, float __phi)
- `long double __gnu_cxx::jacobi_zetal` (long double __k, long double __phi)
- `float __gnu_cxx::jacobif` (unsigned __n, float __alpha, float __beta, float __x)
- `long double __gnu_cxx::jacobil` (unsigned __n, long double __alpha, long double __beta, long double __x)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::type std::laguerre` (unsigned int __n, _Tp __x)
- `float std::laguerref` (unsigned int __n, float __x)
- `long double std::laguerrel` (unsigned int __n, long double __x)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lbincoef` (unsigned int __n, unsigned int __k)
- `float __gnu_cxx::lbincoeff` (unsigned int __n, unsigned int __k)
- `long double __gnu_cxx::lbincoeffl` (unsigned int __n, unsigned int __k)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::ldouble_factorial` (int __n)
- `float __gnu_cxx::ldouble_factorialf` (int __n)
- `long double __gnu_cxx::ldouble_factoriall` (int __n)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::type std::legendre` (unsigned int __l, _Tp __x)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::legendre_q` (unsigned int __n, _Tp __x)
- `float __gnu_cxx::legendre_qf` (unsigned int __n, float __x)
- `long double __gnu_cxx::legendre_ql` (unsigned int __n, long double __x)
- `float std::legendref` (unsigned int __l, float __x)
- `long double std::legendrel` (unsigned int __l, long double __x)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::lfactorial` (unsigned int __n)
- `float __gnu_cxx::lfactorialf` (unsigned int __n)
- `long double __gnu_cxx::lfactoriall` (unsigned int __n)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::logint` (_Tp __x)
- `float __gnu_cxx::logintf` (float __x)
- `long double __gnu_cxx::logintl` (long double __x)
- `template<typename _Tp, typename _Tn >`
`__gnu_cxx::__promote_num_t< _Tp, _Tn > __gnu_cxx::lpochhammer_l` (_Tp __a, _Tn __n)
- `float __gnu_cxx::lpochhammer_lf` (float __a, float __n)

- long double [__gnu_cxx::lpochhammer_ll](#) (long double __a, long double __n)
- template<typename _Tp, typename _Tn >
[__gnu_cxx::__promote_num_t<_Tp, _Tn > __gnu_cxx::lpochhammer_u](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::lpochhammer_uf](#) (float __a, float __n)
- long double [__gnu_cxx::lpochhammer_ul](#) (long double __a, long double __n)
- template<typename _Tph, typename _Tpa >
[__gnu_cxx::__promote_num_t<_Tph, _Tpa > __gnu_cxx::owens_t](#) (_Tph __h, _Tpa __a)
- float [__gnu_cxx::owens_tf](#) (float __h, float __a)
- long double [__gnu_cxx::owens_tl](#) (long double __h, long double __a)
- template<typename _Tp, typename _Tn >
[__gnu_cxx::__promote_num_t<_Tp, _Tn > __gnu_cxx::pochhammer_l](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::pochhammer_lf](#) (float __a, float __n)
- long double [__gnu_cxx::pochhammer_ll](#) (long double __a, long double __n)
- template<typename _Tp, typename _Tn >
[__gnu_cxx::__promote_num_t<_Tp, _Tn > __gnu_cxx::pochhammer_u](#) (_Tp __a, _Tn __n)
- float [__gnu_cxx::pochhammer_uf](#) (float __a, float __n)
- long double [__gnu_cxx::pochhammer_ul](#) (long double __a, long double __n)
- template<typename _Tp, typename _Wp >
[__gnu_cxx::__promote_num_t<_Tp, _Wp > __gnu_cxx::polylog](#) (_Tp __s, _Wp __w)
- template<typename _Tp, typename _Wp >
[std::complex< __gnu_cxx::__promote_num_t<_Tp, _Wp > > __gnu_cxx::polylog](#) (_Tp __s, std::complex<_Tp > __w)
- float [__gnu_cxx::polylogf](#) (float __s, float __w)
- std::complex< float > [__gnu_cxx::polylogf](#) (float __s, std::complex< float > __w)
- long double [__gnu_cxx::polylogl](#) (long double __s, long double __w)
- std::complex< long double > [__gnu_cxx::polylogl](#) (long double __s, std::complex< long double > __w)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::psi](#) (_Tp __x)
- float [__gnu_cxx::psif](#) (float __x)
- long double [__gnu_cxx::psil](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::radpoly](#) (unsigned int __n, unsigned int __m, _Tp __rho)
- float [__gnu_cxx::radpolyf](#) (unsigned int __n, unsigned int __m, float __rho)
- long double [__gnu_cxx::radpolyl](#) (unsigned int __n, unsigned int __m, long double __rho)
- template<typename _Tp >
[__gnu_cxx::__promote<_Tp >::__type std::riemann_zeta](#) (_Tp __s)
- float [std::riemann_zetaf](#) (float __s)
- long double [std::riemann_zetal](#) (long double __s)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sinc](#) (_Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sinc_pi](#) (_Tp __x)
- float [__gnu_cxx::sinc_pif](#) (float __x)
- long double [__gnu_cxx::sinc_pil](#) (long double __x)
- float [__gnu_cxx::sincf](#) (float __x)
- long double [__gnu_cxx::sincl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sinhc](#) (_Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::sinhc_pi](#) (_Tp __x)
- float [__gnu_cxx::sinhc_pif](#) (float __x)
- long double [__gnu_cxx::sinhc_pil](#) (long double __x)

- float [__gnu_cxx::sinhcf](#) (float __x)
- long double [__gnu_cxx::sinhcl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::sinhint](#) (_Tp __x)
- float [__gnu_cxx::sinhintf](#) (float __x)
- long double [__gnu_cxx::sinhintl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::sinint](#) (_Tp __x)
- float [__gnu_cxx::sinintf](#) (float __x)
- long double [__gnu_cxx::sinintl](#) (long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote<_Tp>::__type](#) [std::sph_bessel](#) (unsigned int __n, _Tp __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::sph_bessel_i](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::sph_bessel_if](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::sph_bessel_il](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp>](#) [__gnu_cxx::sph_bessel_k](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::sph_bessel_kf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::sph_bessel_kl](#) (unsigned int __n, long double __x)
- float [std::sph_besself](#) (unsigned int __n, float __x)
- long double [std::sph_bessell](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[std::complex<__gnu_cxx::__promote_num_t<_Tp>>](#) [__gnu_cxx::sph_hankel_1](#) (unsigned int __n, _Tp __z)
- template<typename _Tp >
[std::complex<__gnu_cxx::__promote_num_t<_Tp>>](#) [__gnu_cxx::sph_hankel_1](#) (unsigned int __n, [std::complex<_Tp>](#) __x)
- [std::complex<float>](#) [__gnu_cxx::sph_hankel_1f](#) (unsigned int __n, float __z)
- [std::complex<float>](#) [__gnu_cxx::sph_hankel_1f](#) (unsigned int __n, [std::complex<float>](#) __x)
- [std::complex<long double>](#) [__gnu_cxx::sph_hankel_1l](#) (unsigned int __n, long double __z)
- [std::complex<long double>](#) [__gnu_cxx::sph_hankel_1l](#) (unsigned int __n, [std::complex<long double>](#) __x)
- template<typename _Tp >
[std::complex<__gnu_cxx::__promote_num_t<_Tp>>](#) [__gnu_cxx::sph_hankel_2](#) (unsigned int __n, _Tp __z)
- template<typename _Tp >
[std::complex<__gnu_cxx::__promote_num_t<_Tp>>](#) [__gnu_cxx::sph_hankel_2](#) (unsigned int __n, [std::complex<_Tp>](#) __x)
- [std::complex<float>](#) [__gnu_cxx::sph_hankel_2f](#) (unsigned int __n, float __z)
- [std::complex<float>](#) [__gnu_cxx::sph_hankel_2f](#) (unsigned int __n, [std::complex<float>](#) __x)
- [std::complex<long double>](#) [__gnu_cxx::sph_hankel_2l](#) (unsigned int __n, long double __z)
- [std::complex<long double>](#) [__gnu_cxx::sph_hankel_2l](#) (unsigned int __n, [std::complex<long double>](#) __x)
- template<typename _Ttheta, typename _Tphi >
[std::complex<__gnu_cxx::__promote_num_t<_Ttheta, _Tphi>>](#) [__gnu_cxx::sph_harmonic](#) (unsigned int __l, int __m, _Ttheta __theta, _Tphi __phi)
- [std::complex<float>](#) [__gnu_cxx::sph_harmonicf](#) (unsigned int __l, int __m, float __theta, float __phi)
- [std::complex<long double>](#) [__gnu_cxx::sph_harmonicl](#) (unsigned int __l, int __m, long double __theta, long double __phi)
- template<typename _Tp >
[__gnu_cxx::__promote<_Tp>::__type](#) [std::sph_legendre](#) (unsigned int __l, unsigned int __m, _Tp __theta)
- float [std::sph_legendref](#) (unsigned int __l, unsigned int __m, float __theta)
- long double [std::sph_legendrel](#) (unsigned int __l, unsigned int __m, long double __theta)
- template<typename _Tp >
[__gnu_cxx::__promote<_Tp>::__type](#) [std::sph_neumann](#) (unsigned int __n, _Tp __x)

- float [std::sph_neumannf](#) (unsigned int __n, float __x)
- long double [std::sph_neumannl](#) (unsigned int __n, long double __x)
- template<typename _Tpnu, typename _Tp >
__gnu_cxx::__promote_num_t<_Tpnu, _Tp > [__gnu_cxx::theta_1](#) (_Tpnu __nu, _Tp __x)
- float [__gnu_cxx::theta_1f](#) (float __nu, float __x)
- long double [__gnu_cxx::theta_1l](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
__gnu_cxx::__promote_num_t<_Tpnu, _Tp > [__gnu_cxx::theta_2](#) (_Tpnu __nu, _Tp __x)
- float [__gnu_cxx::theta_2f](#) (float __nu, float __x)
- long double [__gnu_cxx::theta_2l](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
__gnu_cxx::__promote_num_t<_Tpnu, _Tp > [__gnu_cxx::theta_3](#) (_Tpnu __nu, _Tp __x)
- float [__gnu_cxx::theta_3f](#) (float __nu, float __x)
- long double [__gnu_cxx::theta_3l](#) (long double __nu, long double __x)
- template<typename _Tpnu, typename _Tp >
__gnu_cxx::__promote_num_t<_Tpnu, _Tp > [__gnu_cxx::theta_4](#) (_Tpnu __nu, _Tp __x)
- float [__gnu_cxx::theta_4f](#) (float __nu, float __x)
- long double [__gnu_cxx::theta_4l](#) (long double __nu, long double __x)
- template<typename _Tp_k, typename _Tp >
__gnu_cxx::__promote_num_t<_Tp_k, _Tp > [__gnu_cxx::theta_c](#) (_Tp_k __k, _Tp __x)
- float [__gnu_cxx::theta_cf](#) (float __k, float __x)
- long double [__gnu_cxx::theta_cl](#) (long double __k, long double __x)
- template<typename _Tp_k, typename _Tp >
__gnu_cxx::__promote_num_t<_Tp_k, _Tp > [__gnu_cxx::theta_d](#) (_Tp_k __k, _Tp __x)
- float [__gnu_cxx::theta_df](#) (float __k, float __x)
- long double [__gnu_cxx::theta_dl](#) (long double __k, long double __x)
- template<typename _Tp_k, typename _Tp >
__gnu_cxx::__promote_num_t<_Tp_k, _Tp > [__gnu_cxx::theta_n](#) (_Tp_k __k, _Tp __x)
- float [__gnu_cxx::theta_nf](#) (float __k, float __x)
- long double [__gnu_cxx::theta_nl](#) (long double __k, long double __x)
- template<typename _Tp_k, typename _Tp >
__gnu_cxx::__promote_num_t<_Tp_k, _Tp > [__gnu_cxx::theta_s](#) (_Tp_k __k, _Tp __x)
- float [__gnu_cxx::theta_sf](#) (float __k, float __x)
- long double [__gnu_cxx::theta_sl](#) (long double __k, long double __x)
- template<typename _Trho, typename _Tphi >
__gnu_cxx::__promote_num_t<_Trho, _Tphi > [__gnu_cxx::zernike](#) (unsigned int __n, int __m, _Trho __rho, _Tphi __phi)
- float [__gnu_cxx::zernikef](#) (unsigned int __n, int __m, float __rho, float __phi)
- long double [__gnu_cxx::zernikel](#) (unsigned int __n, int __m, long double __rho, long double __phi)

10.27.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

10.27.2 Macro Definition Documentation

10.27.2.1 `#define __cpp_lib_math_special_functions 201603L`

Definition at line 39 of file `specfun.h`.

10.27.2.2 `#define __STDCPP_MATH_SPEC_FUNCS__ 201003L`

Definition at line 37 of file specfun.h.

Index

- `_GLIBCXX_BITS_SF_AIRY_TCC`
 - `sf_airy.tcc`, [237](#)
- `_GLIBCXX_BITS_SF_BESSEL_TCC`
 - `sf_bessel.tcc`, [239](#)
- `_GLIBCXX_BITS_SF_BETA_TCC`
 - `sf_beta.tcc`, [241](#)
- `_GLIBCXX_BITS_SF_CARDINAL_TCC`
 - `sf_cardinal.tcc`, [243](#)
- `_GLIBCXX_BITS_SF_ELLINT_TCC`
 - `sf_ellint.tcc`, [248](#)
- `_GLIBCXX_BITS_SF_EXPINT_TCC`
 - `sf_expint.tcc`, [251](#)
- `_GLIBCXX_BITS_SF_GAMMA_TCC`
 - `sf_gamma.tcc`, [258](#)
- `_GLIBCXX_BITS_SF_HANKEL_NEW_TCC`
 - `sf_hankel_new.tcc`, [262](#)
- `_GLIBCXX_BITS_SF_HANKEL_TCC`
 - `sf_hankel.tcc`, [262](#)
- `_GLIBCXX_BITS_SF_HERMITE_TCC`
 - `sf_hermite.tcc`, [264](#)
- `_GLIBCXX_BITS_SF_HYDROGEN_TCC`
 - `sf_hydrogen.tcc`, [265](#)
- `_GLIBCXX_BITS_SF_HYPERG_TCC`
 - `sf_hyperg.tcc`, [267](#)
- `_GLIBCXX_BITS_SF_LAGUERRE_TCC`
 - `sf_laguerre.tcc`, [272](#)
- `_GLIBCXX_BITS_SF_LEGENDRE_TCC`
 - `sf_legendre.tcc`, [273](#)
- `_GLIBCXX_BITS_SF_MOD_BESSEL_TCC`
 - `sf_mod_bessel.tcc`, [275](#)
- `_GLIBCXX_BITS_SF_OWENS_T_TCC`
 - `sf_owens_t.tcc`, [276](#)
- `_GLIBCXX_BITS_SF_POLYLOG_TCC`
 - `sf_polylog.tcc`, [279](#)
- `_GLIBCXX_BITS_SF_ZETA_TCC`
 - `sf_zeta.tcc`, [285](#)
- `_GLIBCXX_JACOBI_CN`
 - GNU Extended Mathematical Special Functions, [46](#)
- `_GLIBCXX_JACOBI_DN`
 - GNU Extended Mathematical Special Functions, [46](#)
- `_GLIBCXX_JACOBI_SN`
 - GNU Extended Mathematical Special Functions, [46](#)
- `_GLIBCXX_SF_CHEBYSHEV_TCC`
 - `sf_chebyshev.tcc`, [244](#)
- `_GLIBCXX_SF_DAWSON_TCC`
 - `sf_dawson.tcc`, [246](#)
- `_GLIBCXX_SF_FRESNEL_TCC`
 - `sf_fresnel.tcc`, [252](#)
- `_GLIBCXX_SF_GEGENBAUER_TCC`
 - `sf_gegenbauer.tcc`, [259](#)
- `_GLIBCXX_SF_HYPINT_TCC`
 - `sf_hypint.tcc`, [268](#)
- `_GLIBCXX_SF_JACOBI_TCC`
 - `sf_jacobi.tcc`, [270](#)
- `_GLIBCXX_SF_THETA_TCC`
 - `sf_theta.tcc`, [281](#)
- `_GLIBCXX_SF_TRIGINT_TCC`
 - `sf_trigint.tcc`, [283](#)
- `_Num_Euler_Maclaurin_zeta`
 - `std::__detail`, [229](#)
- `_S_Euler_Maclaurin_zeta`
 - `std::__detail`, [229](#)
- `_S_double_factorial_table`
 - `std::__detail`, [229](#)
- `_S_factorial_table`
 - `std::__detail`, [230](#)
- `_S_neg_double_factorial_table`
 - `std::__detail`, [230](#)
- `_S_num_double_factorials`
 - `std::__detail`, [230](#)
- `_S_num_double_factorials< double >`
 - `std::__detail`, [230](#)
- `_S_num_double_factorials< float >`
 - `std::__detail`, [230](#)
- `_S_num_double_factorials< long double >`
 - `std::__detail`, [230](#)
- `_S_num_factorials`
 - `std::__detail`, [230](#)
- `_S_num_factorials< double >`
 - `std::__detail`, [230](#)
- `_S_num_factorials< float >`
 - `std::__detail`, [230](#)
- `_S_num_factorials< long double >`
 - `std::__detail`, [230](#)
- `_S_num_neg_double_factorials`
 - `std::__detail`, [231](#)
- `_S_num_neg_double_factorials< double >`
 - `std::__detail`, [231](#)
- `_S_num_neg_double_factorials< float >`
 - `std::__detail`, [231](#)

- `__S_num_neg_double_factorials< long double >`
std::__detail, 231
- `__S_num_zetam1`
std::__detail, 231
- `__S_zetam1`
std::__detail, 231
- `__STDCPP_MATH_SPEC_FUNCS__`
specfun.h, 296
- `__airy`
std::__detail, 137
- `__airy_ai`
std::__detail, 139
- `__airy_arg`
std::__detail, 139
- `__airy_asymp_absarg_ge_pio3`
std::__detail, 139
- `__airy_asymp_absarg_lt_pio3`
std::__detail, 140
- `__airy_bessel_i`
std::__detail, 140
- `__airy_bessel_k`
std::__detail, 141
- `__airy_bi`
std::__detail, 142
- `__airy_hyperg_rational`
std::__detail, 142
- `__assoc_laguerre`
std::__detail, 143
- `__assoc_legendre_p`
std::__detail, 143
- `__bernoulli`
std::__detail, 144
- `__bernoulli_2n`
std::__detail, 144
- `__bernoulli_series`
std::__detail, 144
- `__beta`
std::__detail, 144
- `__beta_gamma`
std::__detail, 145
- `__beta_inc`
std::__detail, 145
- `__beta_inc_cont_frac`
std::__detail, 146
- `__beta_lgamma`
std::__detail, 146
- `__beta_product`
std::__detail, 146
- `__bincoef`
std::__detail, 147
- `__binomial_cdf`
std::__detail, 147
- `__binomial_cdfc`
std::__detail, 147
- `__bose_einstein`
std::__detail, 148
- `__chebyshev_recur`
std::__detail, 148
- `__chebyshev_t`
std::__detail, 149
- `__chebyshev_u`
std::__detail, 149
- `__chebyshev_v`
std::__detail, 149
- `__chebyshev_w`
std::__detail, 150
- `__chi_squared_pdf`
std::__detail, 150
- `__chi_squared_pdfc`
std::__detail, 150
- `__chshint`
std::__detail, 151
- `__chshint_cont_frac`
std::__detail, 151
- `__chshint_series`
std::__detail, 151
- `__clamp_0_m2pi`
std::__detail, 151
- `__clamp_pi`
std::__detail, 151
- `__clausen`
std::__detail, 152
- `__clausen_c`
std::__detail, 152, 153
- `__clausen_s`
std::__detail, 153
- `__comp_ellint_1`
std::__detail, 154
- `__comp_ellint_2`
std::__detail, 154
- `__comp_ellint_3`
std::__detail, 155
- `__comp_ellint_d`
std::__detail, 155
- `__comp_ellint_rf`
std::__detail, 155
- `__comp_ellint_rg`
std::__detail, 155
- `__conf_hyperg`
std::__detail, 155
- `__conf_hyperg_lim`
std::__detail, 156
- `__conf_hyperg_lim_series`
std::__detail, 156
- `__conf_hyperg_luke`
std::__detail, 156
- `__conf_hyperg_series`
std::__detail, 157

- `__coshint`
 - `std::__detail`, [157](#)
- `__cpp_lib_math_special_functions`
 - `specfun.h`, [296](#)
- `__cyl_bessel`
 - `std::__detail`, [157](#)
- `__cyl_bessel_i`
 - `std::__detail`, [158](#)
- `__cyl_bessel_ij_series`
 - `std::__detail`, [158](#)
- `__cyl_bessel_ik`
 - `std::__detail`, [159](#)
- `__cyl_bessel_ik_asymp`
 - `std::__detail`, [159](#)
- `__cyl_bessel_ik_steel`
 - `std::__detail`, [160](#)
- `__cyl_bessel_j`
 - `std::__detail`, [160](#)
- `__cyl_bessel_jn`
 - `std::__detail`, [160](#)
- `__cyl_bessel_jn_asymp`
 - `std::__detail`, [161](#)
- `__cyl_bessel_jn_steel`
 - `std::__detail`, [161](#)
- `__cyl_bessel_k`
 - `std::__detail`, [161](#)
- `__cyl_hankel_1`
 - `std::__detail`, [162](#)
- `__cyl_hankel_2`
 - `std::__detail`, [162](#), [164](#)
- `__cyl_neumann`
 - `std::__detail`, [164](#)
- `__cyl_neumann_n`
 - `std::__detail`, [164](#)
- `__dawson`
 - `std::__detail`, [166](#)
- `__dawson_cont_frac`
 - `std::__detail`, [166](#)
- `__dawson_series`
 - `std::__detail`, [166](#)
- `__debye_region`
 - `std::__detail`, [166](#)
- `__dilog`
 - `std::__detail`, [166](#)
- `__dirichlet_beta`
 - `std::__detail`, [167](#)
- `__dirichlet_eta`
 - `std::__detail`, [167](#), [169](#)
- `__double_factorial`
 - `std::__detail`, [169](#)
- `__ellint_1`
 - `std::__detail`, [169](#)
- `__ellint_2`
 - `std::__detail`, [170](#)
- `__ellint_3`
 - `std::__detail`, [170](#)
- `__ellint_cel`
 - `std::__detail`, [170](#)
- `__ellint_d`
 - `std::__detail`, [171](#)
- `__ellint_el1`
 - `std::__detail`, [171](#)
- `__ellint_el2`
 - `std::__detail`, [171](#)
- `__ellint_el3`
 - `std::__detail`, [171](#)
- `__ellint_rc`
 - `std::__detail`, [171](#)
- `__ellint_rd`
 - `std::__detail`, [172](#)
- `__ellint_rf`
 - `std::__detail`, [172](#)
- `__ellint_rg`
 - `std::__detail`, [173](#)
- `__ellint_rj`
 - `std::__detail`, [173](#)
- `__ellnome`
 - `std::__detail`, [174](#)
- `__ellnome_k`
 - `std::__detail`, [174](#)
- `__ellnome_series`
 - `std::__detail`, [174](#)
- `__expint`
 - `std::__detail`, [174](#), [175](#)
- `__expint_E1`
 - `std::__detail`, [175](#)
- `__expint_E1_asymp`
 - `std::__detail`, [176](#)
- `__expint_E1_series`
 - `std::__detail`, [176](#)
- `__expint_Ei`
 - `std::__detail`, [177](#)
- `__expint_Ei_asymp`
 - `std::__detail`, [177](#)
- `__expint_Ei_series`
 - `std::__detail`, [177](#)
- `__expint_En_cont_frac`
 - `std::__detail`, [178](#)
- `__expint_En_recursion`
 - `std::__detail`, [178](#)
- `__expint_En_series`
 - `std::__detail`, [179](#)
- `__expint_asymp`
 - `std::__detail`, [175](#)
- `__expint_large_n`
 - `std::__detail`, [179](#)
- `__factorial`
 - `std::__detail`, [179](#)

- std::__detail::_Factorial_table, 233
- __fermi_dirac
 - std::__detail, 180
- __fisher_f_cdf
 - std::__detail, 180
- __fisher_f_cdfc
 - std::__detail, 180
- __fock_airy
 - std::__detail, 181
- __fpequal
 - std::__detail, 181
- __fpimag
 - std::__detail, 181, 182
- __fpreal
 - std::__detail, 182
- __fresnel
 - std::__detail, 182
- __fresnel_cont_frac
 - std::__detail, 183
- __fresnel_series
 - std::__detail, 183
- __gamma
 - std::__detail, 183
- __gamma_cont_frac
 - std::__detail, 183
- __gamma_l
 - std::__detail, 183
- __gamma_p
 - std::__detail, 184
- __gamma_q
 - std::__detail, 184
- __gamma_series
 - std::__detail, 184
- __gamma_temme
 - std::__detail, 184
- __gamma_u
 - std::__detail, 185
- __gauss
 - std::__detail, 185
- __gegenbauer_poly
 - std::__detail, 185
- __gnu_cxx, 109
- __hankel
 - std::__detail, 186
- __hankel_debye
 - std::__detail, 186
- __hankel_params
 - std::__detail, 186
- __hankel_uniform
 - std::__detail, 187
- __hankel_uniform_olver
 - std::__detail, 187
- __hankel_uniform_outer
 - std::__detail, 187
- __hankel_uniform_sum
 - std::__detail, 188
- __heuman_lambda
 - std::__detail, 188
- __hurwitz_zeta
 - std::__detail, 189
- __hurwitz_zeta_euler_maclaurin
 - std::__detail, 189
- __hydrogen
 - std::__detail, 190
- __hyperg
 - std::__detail, 190
- __hyperg_luke
 - std::__detail, 190
- __hyperg_reflect
 - std::__detail, 190
- __hyperg_series
 - std::__detail, 191
- __jacobi_sncndn
 - std::__detail, 191
- __jacobi_zeta
 - std::__detail, 191
- __laguerre
 - std::__detail, 192
- __legendre_q
 - std::__detail, 192
- __log_bincoef
 - std::__detail, 192
- __log_double_factorial
 - std::__detail, 193
- __log_factorial
 - std::__detail, 193
 - std::__detail::_Factorial_table, 233
- __log_gamma
 - std::__detail, 193
- __log_gamma_bernoulli
 - std::__detail, 194
- __log_gamma_lanczos
 - std::__detail, 194
- __log_gamma_sign
 - std::__detail, 194
- __log_gamma_spouge
 - std::__detail, 195
- __log_pochhammer_l
 - std::__detail, 195
- __log_pochhammer_u
 - std::__detail, 196
- __logint
 - std::__detail, 196
- __n
 - std::__detail::_Factorial_table, 233
- __owens_t
 - std::__detail, 196
- __pochhammer_l

- `std::__detail`, [197](#)
- `__pochhammer_u`
 - `std::__detail`, [197](#)
- `__poly_hermite`
 - `std::__detail`, [197](#)
- `__poly_hermite_asymp`
 - `std::__detail`, [198](#)
- `__poly_hermite_recursion`
 - `std::__detail`, [198](#)
- `__poly_jacobi`
 - `std::__detail`, [198](#)
- `__poly_laguerre`
 - `std::__detail`, [199](#)
- `__poly_laguerre_hyperg`
 - `std::__detail`, [199](#)
- `__poly_laguerre_large_n`
 - `std::__detail`, [200](#)
- `__poly_laguerre_recursion`
 - `std::__detail`, [200](#)
- `__poly_legendre_p`
 - `std::__detail`, [201](#)
- `__poly_radial_jacobi`
 - `std::__detail`, [202](#)
- `__polylog`
 - `std::__detail`, [202](#)
- `__polylog_exp`
 - `std::__detail`, [203](#)
- `__polylog_exp_asymp`
 - `std::__detail`, [203](#)
- `__polylog_exp_int_neg`
 - `std::__detail`, [204](#)
- `__polylog_exp_int_pos`
 - `std::__detail`, [204](#), [205](#)
- `__polylog_exp_neg`
 - `std::__detail`, [205](#)
- `__polylog_exp_neg_even`
 - `std::__detail`, [207](#)
- `__polylog_exp_neg_odd`
 - `std::__detail`, [207](#)
- `__polylog_exp_negative_real_part`
 - `std::__detail`, [208](#)
- `__polylog_exp_pos`
 - `std::__detail`, [208](#), [209](#)
- `__polylog_exp_real_neg`
 - `std::__detail`, [210](#)
- `__polylog_exp_real_pos`
 - `std::__detail`, [211](#)
- `__psi`
 - `std::__detail`, [211](#), [212](#)
- `__psi_asymp`
 - `std::__detail`, [212](#)
- `__psi_series`
 - `std::__detail`, [212](#)
- `__riemann_zeta`
 - `std::__detail`, [212](#)
- `__riemann_zeta_alt`
 - `std::__detail`, [213](#)
- `__riemann_zeta_euler_maclaurin`
 - `std::__detail`, [213](#)
- `__riemann_zeta_glob`
 - `std::__detail`, [213](#)
- `__riemann_zeta_m_1`
 - `std::__detail`, [214](#)
- `__riemann_zeta_m_1_sum`
 - `std::__detail`, [214](#)
- `__riemann_zeta_product`
 - `std::__detail`, [214](#)
- `__riemann_zeta_sum`
 - `std::__detail`, [215](#)
- `__sinc`
 - `std::__detail`, [215](#)
- `__sinc_pi`
 - `std::__detail`, [215](#)
- `__sincosint`
 - `std::__detail`, [215](#)
- `__sincosint_asymp`
 - `std::__detail`, [216](#)
- `__sincosint_cont_frac`
 - `std::__detail`, [216](#)
- `__sincosint_series`
 - `std::__detail`, [216](#)
- `__sinhc`
 - `std::__detail`, [216](#)
- `__sinhc_pi`
 - `std::__detail`, [217](#)
- `__sinhint`
 - `std::__detail`, [217](#)
- `__sph_bessel`
 - `std::__detail`, [217](#)
- `__sph_bessel_ik`
 - `std::__detail`, [219](#)
- `__sph_bessel_jn`
 - `std::__detail`, [219](#)
- `__sph_hankel`
 - `std::__detail`, [219](#)
- `__sph_hankel_1`
 - `std::__detail`, [221](#)
- `__sph_hankel_2`
 - `std::__detail`, [221](#), [222](#)
- `__sph_harmonic`
 - `std::__detail`, [222](#)
- `__sph_legendre`
 - `std::__detail`, [223](#)
- `__sph_neumann`
 - `std::__detail`, [223](#)
- `__student_t_cdf`
 - `std::__detail`, [225](#)
- `__student_t_cdfc`

- std::__detail, 225
- __theta_1
 - std::__detail, 225
- __theta_2
 - std::__detail, 226
- __theta_2_asymp
 - std::__detail, 226
- __theta_2_sum
 - std::__detail, 226
- __theta_3
 - std::__detail, 226
- __theta_3_asymp
 - std::__detail, 227
- __theta_3_sum
 - std::__detail, 227
- __theta_4
 - std::__detail, 227
- __theta_c
 - std::__detail, 227
- __theta_d
 - std::__detail, 227
- __theta_n
 - std::__detail, 228
- __theta_s
 - std::__detail, 228
- __zernike
 - std::__detail, 228
- __znorm1
 - std::__detail, 229
- __znorm2
 - std::__detail, 229
- airy_ai
 - GNU Extended Mathematical Special Functions, 46
- airy_aif
 - GNU Extended Mathematical Special Functions, 47
- airy_ail
 - GNU Extended Mathematical Special Functions, 47
- airy_bi
 - GNU Extended Mathematical Special Functions, 47
- airy_bif
 - GNU Extended Mathematical Special Functions, 47
- airy_bil
 - GNU Extended Mathematical Special Functions, 47
- assoc_laguerre
 - C++17/IS29124 Mathematical Special Functions, 18
- assoc_laguerref
 - C++17/IS29124 Mathematical Special Functions, 19
- assoc_laguerrel
 - C++17/IS29124 Mathematical Special Functions, 19
- assoc_legendre
 - C++17/IS29124 Mathematical Special Functions, 19
- assoc_legendref
 - C++17/IS29124 Mathematical Special Functions, 20
- assoc_legendrel
 - C++17/IS29124 Mathematical Special Functions, 20
- bernoulli
 - GNU Extended Mathematical Special Functions, 48
- bernoullif
 - GNU Extended Mathematical Special Functions, 48
- bernoullil
 - GNU Extended Mathematical Special Functions, 48
- beta
 - C++17/IS29124 Mathematical Special Functions, 20
- betaf
 - C++17/IS29124 Mathematical Special Functions, 20
- betal
 - C++17/IS29124 Mathematical Special Functions, 21
- bincoef
 - GNU Extended Mathematical Special Functions, 48
- bincoeff
 - GNU Extended Mathematical Special Functions, 48
- bincoefl
 - GNU Extended Mathematical Special Functions, 48
- bits/sf_airy.tcc, 235
- bits/sf_bessel.tcc, 237
- bits/sf_beta.tcc, 239
- bits/sf_cardinal.tcc, 241
- bits/sf_chebyshev.tcc, 243
- bits/sf_dawson.tcc, 245
- bits/sf_ellint.tcc, 246
- bits/sf_expint.tcc, 249
- bits/sf_fresnel.tcc, 251
- bits/sf_gamma.tcc, 252
- bits/sf_gegenbauer.tcc, 258
- bits/sf_hankel.tcc, 259
- bits/sf_hankel_new.tcc, 262
- bits/sf_hermite.tcc, 263
- bits/sf_hydrogen.tcc, 264
- bits/sf_hyperg.tcc, 265
- bits/sf_hypint.tcc, 267
- bits/sf_jacobi.tcc, 269
- bits/sf_laguerre.tcc, 270
- bits/sf_legendre.tcc, 272
- bits/sf_mod_bessel.tcc, 273
- bits/sf_owens_t.tcc, 276
- bits/sf_polylog.tcc, 277
- bits/sf_theta.tcc, 279
- bits/sf_trigint.tcc, 282
- bits/sf_zeta.tcc, 283
- bits/specfun.h, 286
- C++ Mathematical Special Functions, 15
- C++17/IS29124 Mathematical Special Functions, 16
 - assoc_laguerre, 18
 - assoc_laguerref, 19
 - assoc_laguerrel, 19
 - assoc_legendre, 19

- assoc_legendref, [20](#)
- assoc_legendrel, [20](#)
- beta, [20](#)
- betaf, [20](#)
- betal, [21](#)
- comp_ellint_1, [21](#)
- comp_ellint_1f, [21](#)
- comp_ellint_1l, [21](#)
- comp_ellint_2, [22](#)
- comp_ellint_2f, [22](#)
- comp_ellint_2l, [22](#)
- comp_ellint_3, [22](#)
- comp_ellint_3f, [23](#)
- comp_ellint_3l, [23](#)
- cyl_bessel_i, [23](#)
- cyl_bessel_if, [24](#)
- cyl_bessel_il, [24](#)
- cyl_bessel_j, [24](#)
- cyl_bessel_jf, [25](#)
- cyl_bessel_jl, [25](#)
- cyl_bessel_k, [25](#)
- cyl_bessel_kf, [26](#)
- cyl_bessel_kl, [26](#)
- cyl_neumann, [26](#)
- cyl_neumannf, [27](#)
- cyl_neumannl, [27](#)
- ellint_1, [27](#)
- ellint_1f, [28](#)
- ellint_1l, [28](#)
- ellint_2, [28](#)
- ellint_2f, [29](#)
- ellint_2l, [29](#)
- ellint_3, [29](#)
- ellint_3f, [30](#)
- ellint_3l, [30](#)
- expint, [30](#)
- expintf, [31](#)
- expintl, [31](#)
- hermite, [31](#)
- hermitef, [32](#)
- hermitel, [32](#)
- laguerre, [32](#)
- laguerref, [32](#)
- laguerrel, [33](#)
- legendre, [33](#)
- legendref, [33](#)
- legendrel, [33](#)
- riemann_zeta, [34](#)
- riemann_zetaf, [34](#)
- riemann_zetal, [34](#)
- sph_bessel, [34](#)
- sph_besself, [35](#)
- sph_bessell, [35](#)
- sph_legendre, [35](#)
- sph_legendref, [36](#)
- sph_legendrel, [36](#)
- sph_neumann, [36](#)
- sph_neumannf, [37](#)
- sph_neumannl, [37](#)
- COSINT
 - std::__detail, [137](#)
- chebyshev_t
 - GNU Extended Mathematical Special Functions, [49](#)
- chebyshev_tf
 - GNU Extended Mathematical Special Functions, [49](#)
- chebyshev_tl
 - GNU Extended Mathematical Special Functions, [49](#)
- chebyshev_u
 - GNU Extended Mathematical Special Functions, [49](#)
- chebyshev_uf
 - GNU Extended Mathematical Special Functions, [50](#)
- chebyshev_ul
 - GNU Extended Mathematical Special Functions, [50](#)
- chebyshev_v
 - GNU Extended Mathematical Special Functions, [50](#)
- chebyshev_vf
 - GNU Extended Mathematical Special Functions, [50](#)
- chebyshev_vl
 - GNU Extended Mathematical Special Functions, [51](#)
- chebyshev_w
 - GNU Extended Mathematical Special Functions, [51](#)
- chebyshev_wf
 - GNU Extended Mathematical Special Functions, [51](#)
- chebyshev_wl
 - GNU Extended Mathematical Special Functions, [51](#)
- clausen
 - GNU Extended Mathematical Special Functions, [52](#)
- clausen_c
 - GNU Extended Mathematical Special Functions, [52](#)
- clausen_cf
 - GNU Extended Mathematical Special Functions, [52](#)
- clausen_cl
 - GNU Extended Mathematical Special Functions, [52](#)
- clausen_s
 - GNU Extended Mathematical Special Functions, [53](#)
- clausen_sf
 - GNU Extended Mathematical Special Functions, [53](#)
- clausen_sl
 - GNU Extended Mathematical Special Functions, [53](#)
- clausenf
 - GNU Extended Mathematical Special Functions, [53](#)
- clausenl
 - GNU Extended Mathematical Special Functions, [54](#)
- comp_ellint_1
 - C++17/IS29124 Mathematical Special Functions, [21](#)
- comp_ellint_1f
 - C++17/IS29124 Mathematical Special Functions, [21](#)
- comp_ellint_1l

- C++17/IS29124 Mathematical Special Functions, [21](#)
- comp_ellint_2
 - C++17/IS29124 Mathematical Special Functions, [22](#)
- comp_ellint_2f
 - C++17/IS29124 Mathematical Special Functions, [22](#)
- comp_ellint_2l
 - C++17/IS29124 Mathematical Special Functions, [22](#)
- comp_ellint_3
 - C++17/IS29124 Mathematical Special Functions, [22](#)
- comp_ellint_3f
 - C++17/IS29124 Mathematical Special Functions, [23](#)
- comp_ellint_3l
 - C++17/IS29124 Mathematical Special Functions, [23](#)
- comp_ellint_d
 - GNU Extended Mathematical Special Functions, [54](#)
- comp_ellint_df
 - GNU Extended Mathematical Special Functions, [54](#)
- comp_ellint_dl
 - GNU Extended Mathematical Special Functions, [54](#)
- comp_ellint_rf
 - GNU Extended Mathematical Special Functions, [55](#)
- comp_ellint_rg
 - GNU Extended Mathematical Special Functions, [55](#), [56](#)
- conf_hyperg
 - GNU Extended Mathematical Special Functions, [56](#)
- conf_hyperg_lim
 - GNU Extended Mathematical Special Functions, [56](#)
- conf_hyperg_limf
 - GNU Extended Mathematical Special Functions, [58](#)
- conf_hyperg_liml
 - GNU Extended Mathematical Special Functions, [58](#)
- conf_hypergf
 - GNU Extended Mathematical Special Functions, [58](#)
- conf_hypergl
 - GNU Extended Mathematical Special Functions, [58](#)
- coshint
 - GNU Extended Mathematical Special Functions, [58](#)
- coshintf
 - GNU Extended Mathematical Special Functions, [59](#)
- coshintl
 - GNU Extended Mathematical Special Functions, [59](#)
- cosint
 - GNU Extended Mathematical Special Functions, [59](#)
- cosintf
 - GNU Extended Mathematical Special Functions, [60](#)
- cosintl
 - GNU Extended Mathematical Special Functions, [60](#)
- cyl_bessel_i
 - C++17/IS29124 Mathematical Special Functions, [23](#)
- cyl_bessel_if
 - C++17/IS29124 Mathematical Special Functions, [24](#)
- cyl_bessel_il
 - C++17/IS29124 Mathematical Special Functions, [24](#)
- cyl_bessel_j
 - C++17/IS29124 Mathematical Special Functions, [24](#)
- cyl_bessel_jf
 - C++17/IS29124 Mathematical Special Functions, [25](#)
- cyl_bessel_jl
 - C++17/IS29124 Mathematical Special Functions, [25](#)
- cyl_bessel_k
 - C++17/IS29124 Mathematical Special Functions, [25](#)
- cyl_bessel_kf
 - C++17/IS29124 Mathematical Special Functions, [26](#)
- cyl_bessel_kl
 - C++17/IS29124 Mathematical Special Functions, [26](#)
- cyl_hankel_1
 - GNU Extended Mathematical Special Functions, [60](#)
- cyl_hankel_1f
 - GNU Extended Mathematical Special Functions, [61](#)
- cyl_hankel_1l
 - GNU Extended Mathematical Special Functions, [61](#)
- cyl_hankel_2
 - GNU Extended Mathematical Special Functions, [62](#)
- cyl_hankel_2f
 - GNU Extended Mathematical Special Functions, [63](#)
- cyl_hankel_2l
 - GNU Extended Mathematical Special Functions, [63](#)
- cyl_neumann
 - C++17/IS29124 Mathematical Special Functions, [26](#)
- cyl_neumannf
 - C++17/IS29124 Mathematical Special Functions, [27](#)
- cyl_neumannl
 - C++17/IS29124 Mathematical Special Functions, [27](#)
- dawson
 - GNU Extended Mathematical Special Functions, [63](#)
- dawsonf
 - GNU Extended Mathematical Special Functions, [64](#)
- dawsonl
 - GNU Extended Mathematical Special Functions, [64](#)
- digamma
 - GNU Extended Mathematical Special Functions, [64](#)
- digammaf
 - GNU Extended Mathematical Special Functions, [64](#)
- digammal
 - GNU Extended Mathematical Special Functions, [64](#)
- dilog
 - GNU Extended Mathematical Special Functions, [64](#)
- dilogf
 - GNU Extended Mathematical Special Functions, [65](#)
- dilogl
 - GNU Extended Mathematical Special Functions, [65](#)
- dirichlet_beta
 - GNU Extended Mathematical Special Functions, [65](#)
- dirichlet_betaf
 - GNU Extended Mathematical Special Functions, [65](#)
- dirichlet_betal
 - GNU Extended Mathematical Special Functions, [65](#)

- GNU Extended Mathematical Special Functions, [66](#)
- `dirichlet_eta`
 - GNU Extended Mathematical Special Functions, [66](#)
- `dirichlet_etaf`
 - GNU Extended Mathematical Special Functions, [66](#)
- `dirichlet_etat`
 - GNU Extended Mathematical Special Functions, [66](#)
- `double_factorial`
 - GNU Extended Mathematical Special Functions, [67](#)
- `double_factorialf`
 - GNU Extended Mathematical Special Functions, [67](#)
- `double_factoriall`
 - GNU Extended Mathematical Special Functions, [67](#)
- `ellint_1`
 - C++17/IS29124 Mathematical Special Functions, [27](#)
- `ellint_1f`
 - C++17/IS29124 Mathematical Special Functions, [28](#)
- `ellint_1l`
 - C++17/IS29124 Mathematical Special Functions, [28](#)
- `ellint_2`
 - C++17/IS29124 Mathematical Special Functions, [28](#)
- `ellint_2f`
 - C++17/IS29124 Mathematical Special Functions, [29](#)
- `ellint_2l`
 - C++17/IS29124 Mathematical Special Functions, [29](#)
- `ellint_3`
 - C++17/IS29124 Mathematical Special Functions, [29](#)
- `ellint_3f`
 - C++17/IS29124 Mathematical Special Functions, [30](#)
- `ellint_3l`
 - C++17/IS29124 Mathematical Special Functions, [30](#)
- `ellint_cel`
 - GNU Extended Mathematical Special Functions, [67](#)
- `ellint_celf`
 - GNU Extended Mathematical Special Functions, [67](#)
- `ellint_cell`
 - GNU Extended Mathematical Special Functions, [67](#)
- `ellint_d`
 - GNU Extended Mathematical Special Functions, [68](#)
- `ellint_df`
 - GNU Extended Mathematical Special Functions, [68](#)
- `ellint_dl`
 - GNU Extended Mathematical Special Functions, [68](#)
- `ellint_el1`
 - GNU Extended Mathematical Special Functions, [68](#)
- `ellint_el1f`
 - GNU Extended Mathematical Special Functions, [69](#)
- `ellint_el1l`
 - GNU Extended Mathematical Special Functions, [69](#)
- `ellint_el2`
 - GNU Extended Mathematical Special Functions, [69](#)
- `ellint_el2f`
 - GNU Extended Mathematical Special Functions, [69](#)
- `ellint_el2l`
 - GNU Extended Mathematical Special Functions, [70](#)
- `ellint_el3`
 - GNU Extended Mathematical Special Functions, [70](#)
- `ellint_el3f`
 - GNU Extended Mathematical Special Functions, [70](#)
- `ellint_el3l`
 - GNU Extended Mathematical Special Functions, [70](#)
- `ellint_rc`
 - GNU Extended Mathematical Special Functions, [71](#)
- `ellint_rcf`
 - GNU Extended Mathematical Special Functions, [71](#)
- `ellint_rcl`
 - GNU Extended Mathematical Special Functions, [71](#)
- `ellint_rd`
 - GNU Extended Mathematical Special Functions, [71](#)
- `ellint_rdf`
 - GNU Extended Mathematical Special Functions, [72](#)
- `ellint_rdl`
 - GNU Extended Mathematical Special Functions, [72](#)
- `ellint_rf`
 - GNU Extended Mathematical Special Functions, [72](#)
- `ellint_rff`
 - GNU Extended Mathematical Special Functions, [73](#)
- `ellint_rfl`
 - GNU Extended Mathematical Special Functions, [73](#)
- `ellint_rg`
 - GNU Extended Mathematical Special Functions, [73](#)
- `ellint_rgf`
 - GNU Extended Mathematical Special Functions, [74](#)
- `ellint_rgl`
 - GNU Extended Mathematical Special Functions, [74](#)
- `ellint_rj`
 - GNU Extended Mathematical Special Functions, [74](#)
- `ellint_rjf`
 - GNU Extended Mathematical Special Functions, [75](#)
- `ellint_rjl`
 - GNU Extended Mathematical Special Functions, [75](#)
- `ellnome`
 - GNU Extended Mathematical Special Functions, [75](#)
- `ellnomef`
 - GNU Extended Mathematical Special Functions, [75](#)
- `ellnomel`
 - GNU Extended Mathematical Special Functions, [75](#)
- `evenzeta`
 - `std::__detail`, [229](#)
- `expint`
 - C++17/IS29124 Mathematical Special Functions, [30](#)
- `expint_e1`
 - GNU Extended Mathematical Special Functions, [76](#)
- `expint_e1f`
 - GNU Extended Mathematical Special Functions, [76](#)
- `expint_e1l`
 - GNU Extended Mathematical Special Functions, [76](#)

- expint_en
 - GNU Extended Mathematical Special Functions, 76
- expint_enf
 - GNU Extended Mathematical Special Functions, 76
- expint_enl
 - GNU Extended Mathematical Special Functions, 76
- expintf
 - C++17/IS29124 Mathematical Special Functions, 31
- expintl
 - C++17/IS29124 Mathematical Special Functions, 31
- factorial
 - GNU Extended Mathematical Special Functions, 76
- factorialf
 - GNU Extended Mathematical Special Functions, 76
- factoriall
 - GNU Extended Mathematical Special Functions, 76
- fresnel_c
 - GNU Extended Mathematical Special Functions, 76
- fresnel_cf
 - GNU Extended Mathematical Special Functions, 77
- fresnel_cl
 - GNU Extended Mathematical Special Functions, 77
- fresnel_s
 - GNU Extended Mathematical Special Functions, 77
- fresnel_sf
 - GNU Extended Mathematical Special Functions, 77
- fresnel_sl
 - GNU Extended Mathematical Special Functions, 77
- GNU Extended Mathematical Special Functions, 38
 - _GLIBCXX_JACOBI_CN, 46
 - _GLIBCXX_JACOBI_DN, 46
 - _GLIBCXX_JACOBI_SN, 46
 - airy_ai, 46
 - airy_aif, 47
 - airy_ail, 47
 - airy_bi, 47
 - airy_bif, 47
 - airy_bil, 47
 - bernoulli, 48
 - bernoullif, 48
 - bernoullil, 48
 - bincoef, 48
 - bincoeff, 48
 - bincoefl, 48
 - chebyshev_t, 49
 - chebyshev_tf, 49
 - chebyshev_tl, 49
 - chebyshev_u, 49
 - chebyshev_uf, 50
 - chebyshev_ul, 50
 - chebyshev_v, 50
 - chebyshev_vf, 50
 - chebyshev_vl, 51
 - chebyshev_w, 51
 - chebyshev_wf, 51
 - chebyshev_wl, 51
 - clausen, 52
 - clausen_c, 52
 - clausen_cf, 52
 - clausen_cl, 52
 - clausen_s, 53
 - clausen_sf, 53
 - clausen_sl, 53
 - clausenf, 53
 - clausenl, 54
 - comp_ellint_d, 54
 - comp_ellint_df, 54
 - comp_ellint_dl, 54
 - comp_ellint_rf, 55
 - comp_ellint_rg, 55, 56
 - conf_hyperg, 56
 - conf_hyperg_lim, 56
 - conf_hyperg_limf, 58
 - conf_hyperg_liml, 58
 - conf_hypergf, 58
 - conf_hypergl, 58
 - coshint, 58
 - coshintf, 59
 - coshintl, 59
 - cosint, 59
 - cosintf, 60
 - cosintl, 60
 - cyl_hankel_1, 60
 - cyl_hankel_1f, 61
 - cyl_hankel_1l, 61
 - cyl_hankel_2, 62
 - cyl_hankel_2f, 63
 - cyl_hankel_2l, 63
 - dawson, 63
 - dawsonf, 64
 - dawsonl, 64
 - digamma, 64
 - digammaf, 64
 - digammal, 64
 - dilog, 64
 - dilogf, 65
 - dilogl, 65
 - dirichlet_beta, 65
 - dirichlet_betaf, 65
 - dirichlet_betal, 66
 - dirichlet_eta, 66
 - dirichlet_etaf, 66
 - dirichlet_etall, 66
 - double_factorial, 67
 - double_factorialf, 67
 - double_factoriall, 67
 - ellint_cel, 67

ellint_celf, [67](#)
ellint_cell, [67](#)
ellint_d, [68](#)
ellint_df, [68](#)
ellint_dl, [68](#)
ellint_el1, [68](#)
ellint_el1f, [69](#)
ellint_el1l, [69](#)
ellint_el2, [69](#)
ellint_el2f, [69](#)
ellint_el2l, [70](#)
ellint_el3, [70](#)
ellint_el3f, [70](#)
ellint_el3l, [70](#)
ellint_rc, [71](#)
ellint_rcf, [71](#)
ellint_rcl, [71](#)
ellint_rd, [71](#)
ellint_rdf, [72](#)
ellint_rdl, [72](#)
ellint_rf, [72](#)
ellint_rff, [73](#)
ellint_rfl, [73](#)
ellint_rg, [73](#)
ellint_rgf, [74](#)
ellint_rgl, [74](#)
ellint_rj, [74](#)
ellint_rjf, [75](#)
ellint_rjl, [75](#)
ellnome, [75](#)
ellnomef, [75](#)
ellnomel, [75](#)
expint_e1, [76](#)
expint_e1f, [76](#)
expint_e1l, [76](#)
expint_en, [76](#)
expint_enf, [76](#)
expint_enl, [76](#)
factorial, [76](#)
factorialf, [76](#)
factoriall, [76](#)
fresnel_c, [76](#)
fresnel_cf, [77](#)
fresnel_cl, [77](#)
fresnel_s, [77](#)
fresnel_sf, [77](#)
fresnel_sl, [77](#)
gamma_l, [77](#)
gamma_lf, [77](#)
gamma_ll, [77](#)
gamma_p, [78](#)
gamma_pf, [78](#)
gamma_pl, [78](#)
gamma_q, [78](#)
gamma_qf, [78](#)
gamma_ql, [78](#)
gamma_u, [78](#)
gamma_uf, [78](#)
gamma_ul, [78](#)
gegenbauer, [78](#)
gegenbauerf, [79](#)
gegenbauerl, [79](#)
heuman_lambda, [79](#)
heuman_lambdaf, [80](#)
heuman_lambdal, [80](#)
hurwitz_zeta, [80](#)
hurwitz_zetaf, [80](#)
hurwitz_zetal, [81](#)
hyperg, [81](#)
hypergf, [81](#)
hypergl, [81](#)
ibeta, [82](#)
ibetac, [82](#)
ibetacf, [82](#)
ibetacl, [82](#)
ibetaf, [82](#)
ibetal, [83](#)
jacobi, [83](#)
jacobi_cn, [83](#)
jacobi_cnf, [84](#)
jacobi_cnl, [84](#)
jacobi_dn, [84](#)
jacobi_dnf, [84](#)
jacobi_dnl, [85](#)
jacobi_sn, [85](#)
jacobi_snf, [85](#)
jacobi_snl, [85](#)
jacobi_zeta, [85](#)
jacobi_zetaf, [86](#)
jacobi_zetal, [86](#)
jacobi_f, [86](#)
jacobil, [86](#)
lbincoef, [86](#)
lbincoeff, [87](#)
lbincoefl, [87](#)
ldouble_factorial, [87](#)
ldouble_factorialf, [87](#)
ldouble_factoriall, [87](#)
legendre_q, [87](#)
legendre_qf, [87](#)
legendre_ql, [87](#)
lfactorial, [87](#)
lfactorialf, [88](#)
lfactoriall, [88](#)
logint, [88](#)
logintf, [88](#)
logintl, [88](#)
lpochhammer_l, [88](#)

- lpochhammer_lf, [88](#)
- lpochhammer_ll, [89](#)
- lpochhammer_u, [89](#)
- lpochhammer_uf, [89](#)
- lpochhammer_ul, [89](#)
- owens_t, [89](#)
- owens_tf, [89](#)
- owens_tl, [89](#)
- pochhammer_l, [90](#)
- pochhammer_lf, [90](#)
- pochhammer_ll, [90](#)
- pochhammer_u, [90](#)
- pochhammer_uf, [90](#)
- pochhammer_ul, [90](#)
- polylog, [90](#), [91](#)
- polylogf, [91](#)
- polylogl, [91](#)
- psi, [92](#)
- psif, [92](#)
- psil, [92](#)
- radpoly, [92](#)
- radpolyf, [93](#)
- radpolyl, [93](#)
- sinc, [93](#)
- sinc_pi, [93](#)
- sinc_pif, [93](#)
- sinc_pil, [93](#)
- sincf, [93](#)
- sincl, [94](#)
- sinhc, [94](#)
- sinhc_pi, [94](#)
- sinhc_pif, [94](#)
- sinhc_pil, [94](#)
- sinhcf, [94](#)
- sinhcl, [94](#)
- sinhint, [94](#)
- sinhintf, [95](#)
- sinhintl, [95](#)
- sinint, [95](#)
- sinintf, [95](#)
- sinintl, [95](#)
- sph_bessel_i, [96](#)
- sph_bessel_if, [96](#)
- sph_bessel_il, [96](#)
- sph_bessel_k, [96](#)
- sph_bessel_kf, [97](#)
- sph_bessel_kl, [97](#)
- sph_hankel_1, [97](#), [98](#)
- sph_hankel_1f, [98](#)
- sph_hankel_1l, [98](#), [99](#)
- sph_hankel_2, [99](#)
- sph_hankel_2f, [100](#)
- sph_hankel_2l, [100](#)
- sph_harmonic, [100](#)
- sph_harmonicf, [101](#)
- sph_harmonicl, [101](#)
- theta_1, [101](#)
- theta_1f, [102](#)
- theta_1l, [102](#)
- theta_2, [102](#)
- theta_2f, [102](#)
- theta_2l, [102](#)
- theta_3, [103](#)
- theta_3f, [103](#)
- theta_3l, [103](#)
- theta_4, [103](#)
- theta_4f, [104](#)
- theta_4l, [104](#)
- theta_c, [104](#)
- theta_cf, [104](#)
- theta_cl, [104](#)
- theta_d, [105](#)
- theta_df, [105](#)
- theta_dl, [105](#)
- theta_n, [105](#)
- theta_nf, [106](#)
- theta_nl, [106](#)
- theta_s, [106](#)
- theta_sf, [106](#)
- theta_sl, [106](#)
- zernike, [107](#)
- zernikef, [107](#)
- zernikel, [107](#)
- gamma_l
 - GNU Extended Mathematical Special Functions, [77](#)
- gamma_lf
 - GNU Extended Mathematical Special Functions, [77](#)
- gamma_ll
 - GNU Extended Mathematical Special Functions, [77](#)
- gamma_p
 - GNU Extended Mathematical Special Functions, [78](#)
- gamma_pf
 - GNU Extended Mathematical Special Functions, [78](#)
- gamma_pl
 - GNU Extended Mathematical Special Functions, [78](#)
- gamma_q
 - GNU Extended Mathematical Special Functions, [78](#)
- gamma_qf
 - GNU Extended Mathematical Special Functions, [78](#)
- gamma_ql
 - GNU Extended Mathematical Special Functions, [78](#)
- gamma_u
 - GNU Extended Mathematical Special Functions, [78](#)
- gamma_uf
 - GNU Extended Mathematical Special Functions, [78](#)
- gamma_ul
 - GNU Extended Mathematical Special Functions, [78](#)
- gegenbauer

- GNU Extended Mathematical Special Functions, [78](#)
- gegenbauerf
 - GNU Extended Mathematical Special Functions, [79](#)
- gegenbauerl
 - GNU Extended Mathematical Special Functions, [79](#)
- hermite
 - C++17/IS29124 Mathematical Special Functions, [31](#)
- hermitef
 - C++17/IS29124 Mathematical Special Functions, [32](#)
- hermitel
 - C++17/IS29124 Mathematical Special Functions, [32](#)
- heuman_lambda
 - GNU Extended Mathematical Special Functions, [79](#)
- heuman_lambdaf
 - GNU Extended Mathematical Special Functions, [80](#)
- heuman_lambdal
 - GNU Extended Mathematical Special Functions, [80](#)
- hurwitz_zeta
 - GNU Extended Mathematical Special Functions, [80](#)
- hurwitz_zetaf
 - GNU Extended Mathematical Special Functions, [80](#)
- hurwitz_zetal
 - GNU Extended Mathematical Special Functions, [81](#)
- hyperg
 - GNU Extended Mathematical Special Functions, [81](#)
- hypergf
 - GNU Extended Mathematical Special Functions, [81](#)
- hypergl
 - GNU Extended Mathematical Special Functions, [81](#)
- ibeta
 - GNU Extended Mathematical Special Functions, [82](#)
- ibetac
 - GNU Extended Mathematical Special Functions, [82](#)
- ibetacf
 - GNU Extended Mathematical Special Functions, [82](#)
- ibetacl
 - GNU Extended Mathematical Special Functions, [82](#)
- ibetaf
 - GNU Extended Mathematical Special Functions, [82](#)
- ibetal
 - GNU Extended Mathematical Special Functions, [83](#)
- jacobi
 - GNU Extended Mathematical Special Functions, [83](#)
- jacobi_cn
 - GNU Extended Mathematical Special Functions, [83](#)
- jacobi_cnf
 - GNU Extended Mathematical Special Functions, [84](#)
- jacobi_cnl
 - GNU Extended Mathematical Special Functions, [84](#)
- jacobi_dn
 - GNU Extended Mathematical Special Functions, [84](#)
- jacobi_dnf
 - GNU Extended Mathematical Special Functions, [84](#)
- jacobi_dnl
 - GNU Extended Mathematical Special Functions, [85](#)
- jacobi_sn
 - GNU Extended Mathematical Special Functions, [85](#)
- jacobi_snf
 - GNU Extended Mathematical Special Functions, [85](#)
- jacobi_snl
 - GNU Extended Mathematical Special Functions, [85](#)
- jacobi_zeta
 - GNU Extended Mathematical Special Functions, [85](#)
- jacobi_zetaf
 - GNU Extended Mathematical Special Functions, [86](#)
- jacobi_zetal
 - GNU Extended Mathematical Special Functions, [86](#)
- jacobif
 - GNU Extended Mathematical Special Functions, [86](#)
- jacobil
 - GNU Extended Mathematical Special Functions, [86](#)
- laguerre
 - C++17/IS29124 Mathematical Special Functions, [32](#)
- laguerref
 - C++17/IS29124 Mathematical Special Functions, [32](#)
- laguerrel
 - C++17/IS29124 Mathematical Special Functions, [33](#)
- lbincoef
 - GNU Extended Mathematical Special Functions, [86](#)
- lbincoefficient
 - GNU Extended Mathematical Special Functions, [87](#)
- lbincoefficientl
 - GNU Extended Mathematical Special Functions, [87](#)
- ldouble_factorial
 - GNU Extended Mathematical Special Functions, [87](#)
- ldouble_factorialf
 - GNU Extended Mathematical Special Functions, [87](#)
- ldouble_factoriall
 - GNU Extended Mathematical Special Functions, [87](#)
- legendre
 - C++17/IS29124 Mathematical Special Functions, [33](#)
- legendre_q
 - GNU Extended Mathematical Special Functions, [87](#)
- legendre_qf
 - GNU Extended Mathematical Special Functions, [87](#)
- legendre_ql
 - GNU Extended Mathematical Special Functions, [87](#)
- legendref
 - C++17/IS29124 Mathematical Special Functions, [33](#)
- legendrel
 - C++17/IS29124 Mathematical Special Functions, [33](#)
- lfactorial
 - GNU Extended Mathematical Special Functions, [87](#)
- lfactorialf
 - GNU Extended Mathematical Special Functions, [88](#)

- lfactorial
 - GNU Extended Mathematical Special Functions, [88](#)
- logint
 - GNU Extended Mathematical Special Functions, [88](#)
- logintf
 - GNU Extended Mathematical Special Functions, [88](#)
- logintl
 - GNU Extended Mathematical Special Functions, [88](#)
- lpochhammer_l
 - GNU Extended Mathematical Special Functions, [88](#)
- lpochhammer_lf
 - GNU Extended Mathematical Special Functions, [88](#)
- lpochhammer_ll
 - GNU Extended Mathematical Special Functions, [89](#)
- lpochhammer_u
 - GNU Extended Mathematical Special Functions, [89](#)
- lpochhammer_uf
 - GNU Extended Mathematical Special Functions, [89](#)
- lpochhammer_ul
 - GNU Extended Mathematical Special Functions, [89](#)
- owens_t
 - GNU Extended Mathematical Special Functions, [89](#)
- owens_tf
 - GNU Extended Mathematical Special Functions, [89](#)
- owens_tl
 - GNU Extended Mathematical Special Functions, [89](#)
- pochhammer_l
 - GNU Extended Mathematical Special Functions, [90](#)
- pochhammer_lf
 - GNU Extended Mathematical Special Functions, [90](#)
- pochhammer_ll
 - GNU Extended Mathematical Special Functions, [90](#)
- pochhammer_u
 - GNU Extended Mathematical Special Functions, [90](#)
- pochhammer_uf
 - GNU Extended Mathematical Special Functions, [90](#)
- pochhammer_ul
 - GNU Extended Mathematical Special Functions, [90](#)
- polylog
 - GNU Extended Mathematical Special Functions, [90](#), [91](#)
- polylogf
 - GNU Extended Mathematical Special Functions, [91](#)
- polylogl
 - GNU Extended Mathematical Special Functions, [91](#)
- psi
 - GNU Extended Mathematical Special Functions, [92](#)
- psif
 - GNU Extended Mathematical Special Functions, [92](#)
- psil
 - GNU Extended Mathematical Special Functions, [92](#)
- radpoly
 - GNU Extended Mathematical Special Functions, [92](#)
- radpolyf
 - GNU Extended Mathematical Special Functions, [93](#)
- radpolyl
 - GNU Extended Mathematical Special Functions, [93](#)
- riemann_zeta
 - C++17/IS29124 Mathematical Special Functions, [34](#)
- riemann_zetaf
 - C++17/IS29124 Mathematical Special Functions, [34](#)
- riemann_zetal
 - C++17/IS29124 Mathematical Special Functions, [34](#)
- SININT
 - std::__detail, [137](#)
- sf_airy.tcc
 - _GLIBCXX_BITS_SF_AIRY_TCC, [237](#)
- sf_bessel.tcc
 - _GLIBCXX_BITS_SF_BESSEL_TCC, [239](#)
- sf_beta.tcc
 - _GLIBCXX_BITS_SF_BETA_TCC, [241](#)
- sf_cardinal.tcc
 - _GLIBCXX_BITS_SF_CARDINAL_TCC, [243](#)
- sf_chebyshev.tcc
 - _GLIBCXX_SF_CHEBYSHEV_TCC, [244](#)
- sf_dawson.tcc
 - _GLIBCXX_SF_DAWSON_TCC, [246](#)
- sf_ellint.tcc
 - _GLIBCXX_BITS_SF_ELLINT_TCC, [248](#)
- sf_expint.tcc
 - _GLIBCXX_BITS_SF_EXPINT_TCC, [251](#)
- sf_fresnel.tcc
 - _GLIBCXX_SF_FRESNEL_TCC, [252](#)
- sf_gamma.tcc
 - _GLIBCXX_BITS_SF_GAMMA_TCC, [258](#)
- sf_gegenbauer.tcc
 - _GLIBCXX_SF_GEGENBAUER_TCC, [259](#)
- sf_hankel.tcc
 - _GLIBCXX_BITS_SF_HANKEL_TCC, [262](#)
- sf_hankel_new.tcc
 - _GLIBCXX_BITS_SF_HANKEL_NEW_TCC, [262](#)
- sf_hermite.tcc
 - _GLIBCXX_BITS_SF_HERMITE_TCC, [264](#)
- sf_hydrogen.tcc
 - _GLIBCXX_BITS_SF_HYDROGEN_TCC, [265](#)
- sf_hyperg.tcc
 - _GLIBCXX_BITS_SF_HYPERG_TCC, [267](#)
- sf_hypint.tcc
 - _GLIBCXX_SF_HYPINT_TCC, [268](#)
- sf_jacobi.tcc
 - _GLIBCXX_SF_JACOBI_TCC, [270](#)
- sf_laguerre.tcc
 - _GLIBCXX_BITS_SF_LAGUERRE_TCC, [272](#)
- sf_legendre.tcc
 - _GLIBCXX_BITS_SF_LEGENDRE_TCC, [273](#)

- sf_mod_bessel.tcc
 - _GLIBCXX_BITS_SF_MOD_BESSEL_TCC, [275](#)
- sf_owens.tcc
 - _GLIBCXX_BITS_SF_OWENS_T_TCC, [276](#)
- sf_polylog.tcc
 - _GLIBCXX_BITS_SF_POLYLOG_TCC, [279](#)
- sf_theta.tcc
 - _GLIBCXX_SF_THETA_TCC, [281](#)
- sf_trigint.tcc
 - _GLIBCXX_SF_TRIGINT_TCC, [283](#)
- sf_zeta.tcc
 - _GLIBCXX_BITS_SF_ZETA_TCC, [285](#)
- sinc
 - GNU Extended Mathematical Special Functions, [93](#)
- sinc_pi
 - GNU Extended Mathematical Special Functions, [93](#)
- sinc_pif
 - GNU Extended Mathematical Special Functions, [93](#)
- sinc_pil
 - GNU Extended Mathematical Special Functions, [93](#)
- sincf
 - GNU Extended Mathematical Special Functions, [93](#)
- sincl
 - GNU Extended Mathematical Special Functions, [94](#)
- sinhc
 - GNU Extended Mathematical Special Functions, [94](#)
- sinhc_pi
 - GNU Extended Mathematical Special Functions, [94](#)
- sinhc_pif
 - GNU Extended Mathematical Special Functions, [94](#)
- sinhc_pil
 - GNU Extended Mathematical Special Functions, [94](#)
- sinhcf
 - GNU Extended Mathematical Special Functions, [94](#)
- sinhcl
 - GNU Extended Mathematical Special Functions, [94](#)
- sinhint
 - GNU Extended Mathematical Special Functions, [94](#)
- sinhintf
 - GNU Extended Mathematical Special Functions, [95](#)
- sinhintl
 - GNU Extended Mathematical Special Functions, [95](#)
- sinint
 - GNU Extended Mathematical Special Functions, [95](#)
- sinintf
 - GNU Extended Mathematical Special Functions, [95](#)
- sinintl
 - GNU Extended Mathematical Special Functions, [95](#)
- specfun.h
 - _STDCPP_MATH_SPEC_FUNCS_, [296](#)
 - _cpp_lib_math_special_functions, [296](#)
- sph_bessel
 - C++17/IS29124 Mathematical Special Functions, [34](#)
- sph_bessel_i
 - GNU Extended Mathematical Special Functions, [96](#)
- sph_bessel_if
 - GNU Extended Mathematical Special Functions, [96](#)
- sph_bessel_il
 - GNU Extended Mathematical Special Functions, [96](#)
- sph_bessel_k
 - GNU Extended Mathematical Special Functions, [96](#)
- sph_bessel_kf
 - GNU Extended Mathematical Special Functions, [97](#)
- sph_bessel_kl
 - GNU Extended Mathematical Special Functions, [97](#)
- sph_besself
 - C++17/IS29124 Mathematical Special Functions, [35](#)
- sph_bessell
 - C++17/IS29124 Mathematical Special Functions, [35](#)
- sph_hankel_1
 - GNU Extended Mathematical Special Functions, [97](#), [98](#)
- sph_hankel_1f
 - GNU Extended Mathematical Special Functions, [98](#)
- sph_hankel_1l
 - GNU Extended Mathematical Special Functions, [98](#), [99](#)
- sph_hankel_2
 - GNU Extended Mathematical Special Functions, [99](#)
- sph_hankel_2f
 - GNU Extended Mathematical Special Functions, [100](#)
- sph_hankel_2l
 - GNU Extended Mathematical Special Functions, [100](#)
- sph_harmonic
 - GNU Extended Mathematical Special Functions, [100](#)
- sph_harmonicf
 - GNU Extended Mathematical Special Functions, [101](#)
- sph_harmonicl
 - GNU Extended Mathematical Special Functions, [101](#)
- sph_legendre
 - C++17/IS29124 Mathematical Special Functions, [35](#)
- sph_legendref
 - C++17/IS29124 Mathematical Special Functions, [36](#)
- sph_legendrel
 - C++17/IS29124 Mathematical Special Functions, [36](#)
- sph_neumann
 - C++17/IS29124 Mathematical Special Functions, [36](#)
- sph_neumannf
 - C++17/IS29124 Mathematical Special Functions, [37](#)
- sph_neumannl
 - C++17/IS29124 Mathematical Special Functions, [37](#)
- std, [117](#)
- std::__detail, [119](#)
- _Num_Euler_Maclaurin_zeta, [229](#)
- _S_Euler_Maclaurin_zeta, [229](#)
- _S_double_factorial_table, [229](#)
- _S_factorial_table, [230](#)
- _S_neg_double_factorial_table, [230](#)

- [_S_num_double_factorials](#), 230
- [_S_num_double_factorials< double >](#), 230
- [_S_num_double_factorials< float >](#), 230
- [_S_num_double_factorials< long double >](#), 230
- [_S_num_factorials](#), 230
- [_S_num_factorials< double >](#), 230
- [_S_num_factorials< float >](#), 230
- [_S_num_factorials< long double >](#), 230
- [_S_num_neg_double_factorials](#), 231
- [_S_num_neg_double_factorials< double >](#), 231
- [_S_num_neg_double_factorials< float >](#), 231
- [_S_num_neg_double_factorials< long double >](#), 231
- [_S_num_zetam1](#), 231
- [_S_zetam1](#), 231
- [_airy](#), 137
- [_airy_ai](#), 139
- [_airy_arg](#), 139
- [_airy_asymp_absarg_ge_pio3](#), 139
- [_airy_asymp_absarg_lt_pio3](#), 140
- [_airy_bessel_i](#), 140
- [_airy_bessel_k](#), 141
- [_airy_bi](#), 142
- [_airy_hyperg_rational](#), 142
- [_assoc_laguerre](#), 143
- [_assoc_legendre_p](#), 143
- [_bernoulli](#), 144
- [_bernoulli_2n](#), 144
- [_bernoulli_series](#), 144
- [_beta](#), 144
- [_beta_gamma](#), 145
- [_beta_inc](#), 145
- [_beta_inc_cont_frac](#), 146
- [_beta_lgamma](#), 146
- [_beta_product](#), 146
- [_bincoef](#), 147
- [_binomial_cdf](#), 147
- [_binomial_cdfc](#), 147
- [_bose_einstein](#), 148
- [_chebyshev_recur](#), 148
- [_chebyshev_t](#), 149
- [_chebyshev_u](#), 149
- [_chebyshev_v](#), 149
- [_chebyshev_w](#), 150
- [_chi_squared_pdf](#), 150
- [_chi_squared_pdfc](#), 150
- [_chshint](#), 151
- [_chshint_cont_frac](#), 151
- [_chshint_series](#), 151
- [_clamp_0_m2pi](#), 151
- [_clamp_pi](#), 151
- [_clausen](#), 152
- [_clausen_c](#), 152, 153
- [_clausen_s](#), 153
- [_comp_ellint_1](#), 154
- [_comp_ellint_2](#), 154
- [_comp_ellint_3](#), 155
- [_comp_ellint_d](#), 155
- [_comp_ellint_rf](#), 155
- [_comp_ellint_rg](#), 155
- [_conf_hyperg](#), 155
- [_conf_hyperg_lim](#), 156
- [_conf_hyperg_lim_series](#), 156
- [_conf_hyperg_luke](#), 156
- [_conf_hyperg_series](#), 157
- [_coshint](#), 157
- [_cyl_bessel](#), 157
- [_cyl_bessel_i](#), 158
- [_cyl_bessel_ij_series](#), 158
- [_cyl_bessel_ik](#), 159
- [_cyl_bessel_ik_asymp](#), 159
- [_cyl_bessel_ik_steel](#), 160
- [_cyl_bessel_j](#), 160
- [_cyl_bessel_jn](#), 160
- [_cyl_bessel_jn_asymp](#), 161
- [_cyl_bessel_jn_steel](#), 161
- [_cyl_bessel_k](#), 161
- [_cyl_hankel_1](#), 162
- [_cyl_hankel_2](#), 162, 164
- [_cyl_neumann](#), 164
- [_cyl_neumann_n](#), 164
- [_dawson](#), 166
- [_dawson_cont_frac](#), 166
- [_dawson_series](#), 166
- [_debye_region](#), 166
- [_dilog](#), 166
- [_dirichlet_beta](#), 167
- [_dirichlet_eta](#), 167, 169
- [_double_factorial](#), 169
- [_ellint_1](#), 169
- [_ellint_2](#), 170
- [_ellint_3](#), 170
- [_ellint_cel](#), 170
- [_ellint_d](#), 171
- [_ellint_el1](#), 171
- [_ellint_el2](#), 171
- [_ellint_el3](#), 171
- [_ellint_rc](#), 171
- [_ellint_rd](#), 172
- [_ellint_rf](#), 172
- [_ellint_rg](#), 173
- [_ellint_rj](#), 173
- [_ellnome](#), 174
- [_ellnome_k](#), 174
- [_ellnome_series](#), 174
- [_expint](#), 174, 175
- [_expint_E1](#), 175
- [_expint_E1_asymp](#), 176
- [_expint_E1_series](#), 176

- [__expint_Ei, 177](#)
- [__expint_Ei_asymp, 177](#)
- [__expint_Ei_series, 177](#)
- [__expint_En_cont_frac, 178](#)
- [__expint_En_recursion, 178](#)
- [__expint_En_series, 179](#)
- [__expint_asymp, 175](#)
- [__expint_large_n, 179](#)
- [__factorial, 179](#)
- [__fermi_dirac, 180](#)
- [__fisher_f_cdf, 180](#)
- [__fisher_f_cdfc, 180](#)
- [__fock_airy, 181](#)
- [__fpequal, 181](#)
- [__fpimag, 181, 182](#)
- [__fpreal, 182](#)
- [__fresnel, 182](#)
- [__fresnel_cont_frac, 183](#)
- [__fresnel_series, 183](#)
- [__gamma, 183](#)
- [__gamma_cont_frac, 183](#)
- [__gamma_l, 183](#)
- [__gamma_p, 184](#)
- [__gamma_q, 184](#)
- [__gamma_series, 184](#)
- [__gamma_temme, 184](#)
- [__gamma_u, 185](#)
- [__gauss, 185](#)
- [__gegenbauer_poly, 185](#)
- [__hankel, 186](#)
- [__hankel_debye, 186](#)
- [__hankel_params, 186](#)
- [__hankel_uniform, 187](#)
- [__hankel_uniform_olver, 187](#)
- [__hankel_uniform_outer, 187](#)
- [__hankel_uniform_sum, 188](#)
- [__heuman_lambda, 188](#)
- [__hurwitz_zeta, 189](#)
- [__hurwitz_zeta_euler_maclaurin, 189](#)
- [__hydrogen, 190](#)
- [__hyperg, 190](#)
- [__hyperg_luke, 190](#)
- [__hyperg_reflect, 190](#)
- [__hyperg_series, 191](#)
- [__jacobi_sncndn, 191](#)
- [__jacobi_zeta, 191](#)
- [__laguerre, 192](#)
- [__legendre_q, 192](#)
- [__log_bincoef, 192](#)
- [__log_double_factorial, 193](#)
- [__log_factorial, 193](#)
- [__log_gamma, 193](#)
- [__log_gamma_bernoulli, 194](#)
- [__log_gamma_lanczos, 194](#)
- [__log_gamma_sign, 194](#)
- [__log_gamma_spouge, 195](#)
- [__log_pochhammer_l, 195](#)
- [__log_pochhammer_u, 196](#)
- [__logint, 196](#)
- [__owens_t, 196](#)
- [__pochhammer_l, 197](#)
- [__pochhammer_u, 197](#)
- [__poly_hermite, 197](#)
- [__poly_hermite_asymp, 198](#)
- [__poly_hermite_recursion, 198](#)
- [__poly_jacobi, 198](#)
- [__poly_laguerre, 199](#)
- [__poly_laguerre_hyperg, 199](#)
- [__poly_laguerre_large_n, 200](#)
- [__poly_laguerre_recursion, 200](#)
- [__poly_legendre_p, 201](#)
- [__poly_radial_jacobi, 202](#)
- [__polylog, 202](#)
- [__polylog_exp, 203](#)
- [__polylog_exp_asymp, 203](#)
- [__polylog_exp_int_neg, 204](#)
- [__polylog_exp_int_pos, 204, 205](#)
- [__polylog_exp_neg, 205](#)
- [__polylog_exp_neg_even, 207](#)
- [__polylog_exp_neg_odd, 207](#)
- [__polylog_exp_negative_real_part, 208](#)
- [__polylog_exp_pos, 208, 209](#)
- [__polylog_exp_real_neg, 210](#)
- [__polylog_exp_real_pos, 211](#)
- [__psi, 211, 212](#)
- [__psi_asymp, 212](#)
- [__psi_series, 212](#)
- [__riemann_zeta, 212](#)
- [__riemann_zeta_alt, 213](#)
- [__riemann_zeta_euler_maclaurin, 213](#)
- [__riemann_zeta_glob, 213](#)
- [__riemann_zeta_m_1, 214](#)
- [__riemann_zeta_m_1_sum, 214](#)
- [__riemann_zeta_product, 214](#)
- [__riemann_zeta_sum, 215](#)
- [__sinc, 215](#)
- [__sinc_pi, 215](#)
- [__sincosint, 215](#)
- [__sincosint_asymp, 216](#)
- [__sincosint_cont_frac, 216](#)
- [__sincosint_series, 216](#)
- [__sinhc, 216](#)
- [__sinhc_pi, 217](#)
- [__sinhint, 217](#)
- [__sph_bessel, 217](#)
- [__sph_bessel_ik, 219](#)
- [__sph_bessel_jn, 219](#)
- [__sph_hankel, 219](#)

- __sph_hankel_1, 221
- __sph_hankel_2, 221, 222
- __sph_harmonic, 222
- __sph_legendre, 223
- __sph_neumann, 223
- __student_t_cdf, 225
- __student_t_cdfc, 225
- __theta_1, 225
- __theta_2, 226
- __theta_2_asyp, 226
- __theta_2_sum, 226
- __theta_3, 226
- __theta_3_asyp, 227
- __theta_3_sum, 227
- __theta_4, 227
- __theta_c, 227
- __theta_d, 227
- __theta_n, 228
- __theta_s, 228
- __zernike, 228
- __znorm1, 229
- __znorm2, 229
- COSINT, 137
- evenzeta, 229
- SININT, 137
- std::__detail::_Factorial_table
 - __factorial, 233
 - __log_factorial, 233
 - __n, 233
- std::__detail::_Factorial_table< _Tp >, 233
- theta_1
 - GNU Extended Mathematical Special Functions, 101
- theta_1f
 - GNU Extended Mathematical Special Functions, 102
- theta_1l
 - GNU Extended Mathematical Special Functions, 102
- theta_2
 - GNU Extended Mathematical Special Functions, 102
- theta_2f
 - GNU Extended Mathematical Special Functions, 102
- theta_2l
 - GNU Extended Mathematical Special Functions, 102
- theta_3
 - GNU Extended Mathematical Special Functions, 103
- theta_3f
 - GNU Extended Mathematical Special Functions, 103
- theta_3l
 - GNU Extended Mathematical Special Functions, 103
- theta_4
 - GNU Extended Mathematical Special Functions, 103
- theta_4f
 - GNU Extended Mathematical Special Functions, 104
- theta_4l
 - GNU Extended Mathematical Special Functions, 104
- theta_c
 - GNU Extended Mathematical Special Functions, 104
- theta_cf
 - GNU Extended Mathematical Special Functions, 104
- theta_cl
 - GNU Extended Mathematical Special Functions, 104
- theta_d
 - GNU Extended Mathematical Special Functions, 105
- theta_df
 - GNU Extended Mathematical Special Functions, 105
- theta_dl
 - GNU Extended Mathematical Special Functions, 105
- theta_n
 - GNU Extended Mathematical Special Functions, 105
- theta_nf
 - GNU Extended Mathematical Special Functions, 106
- theta_nl
 - GNU Extended Mathematical Special Functions, 106
- theta_s
 - GNU Extended Mathematical Special Functions, 106
- theta_sf
 - GNU Extended Mathematical Special Functions, 106
- theta_sl
 - GNU Extended Mathematical Special Functions, 106
- zernike
 - GNU Extended Mathematical Special Functions, 107
- zernikef
 - GNU Extended Mathematical Special Functions, 107
- zernikel
 - GNU Extended Mathematical Special Functions, 107