

TR29124 C++ Special Math Functions

2.0

Generated by Doxygen 1.8.9.1

Wed Apr 6 2016 13:04:22

Contents

1	Todo List	1
2	Module Index	3
2.1	Modules	3
3	Namespace Index	5
3.1	Namespace List	5
4	Class Index	7
4.1	Class List	7
5	File Index	9
5.1	File List	9
6	Module Documentation	11
6.1	Extended Mathematical Special Functions	11
6.1.1	Detailed Description	19
6.1.2	Enumeration Type Documentation	19
6.1.2.1	anonymous enum	19
6.1.3	Function Documentation	19
6.1.3.1	airy_ai	19
6.1.3.2	airy_aif	19
6.1.3.3	airy_ail	19
6.1.3.4	airy_bi	19
6.1.3.5	airy_bif	20
6.1.3.6	airy_bil	20
6.1.3.7	bernoulli	20
6.1.3.8	bernoullif	20
6.1.3.9	bernoullil	20
6.1.3.10	bincoef	20

6.1.3.11	bincoeff	20
6.1.3.12	bincoefl	20
6.1.3.13	chebyshev_t	20
6.1.3.14	chebyshev_tf	21
6.1.3.15	chebyshev_tl	21
6.1.3.16	chebyshev_u	21
6.1.3.17	chebyshev_uf	21
6.1.3.18	chebyshev_ul	22
6.1.3.19	chebyshev_v	22
6.1.3.20	chebyshev_vf	22
6.1.3.21	chebyshev_vl	22
6.1.3.22	chebyshev_w	22
6.1.3.23	chebyshev_wf	23
6.1.3.24	chebyshev_wl	23
6.1.3.25	clausen	23
6.1.3.26	clausen	23
6.1.3.27	clausen_c	23
6.1.3.28	clausen_cf	24
6.1.3.29	clausen_cl	24
6.1.3.30	clausen_s	24
6.1.3.31	clausen_sf	24
6.1.3.32	clausen_sl	25
6.1.3.33	clausenf	25
6.1.3.34	clausenf	25
6.1.3.35	clausenl	25
6.1.3.36	clausenl	25
6.1.3.37	comp_ellint_d	25
6.1.3.38	comp_ellint_df	26
6.1.3.39	comp_ellint_dl	26
6.1.3.40	comp_ellint_rf	26
6.1.3.41	comp_ellint_rf	26
6.1.3.42	comp_ellint_rf	26
6.1.3.43	comp_ellint_rg	26
6.1.3.44	comp_ellint_rg	26
6.1.3.45	comp_ellint_rg	27
6.1.3.46	conf_hyperg	27
6.1.3.47	conf_hyperg_lim	27

6.1.3.48	conf_hyperg_limf	27
6.1.3.49	conf_hyperg_liml	28
6.1.3.50	conf_hypergf	28
6.1.3.51	conf_hypergl	28
6.1.3.52	coshint	28
6.1.3.53	coshintf	28
6.1.3.54	coshintl	29
6.1.3.55	cosint	29
6.1.3.56	cosintf	29
6.1.3.57	cosintl	29
6.1.3.58	cyl_hankel_1	29
6.1.3.59	cyl_hankel_1	29
6.1.3.60	cyl_hankel_1f	31
6.1.3.61	cyl_hankel_1f	31
6.1.3.62	cyl_hankel_1l	31
6.1.3.63	cyl_hankel_1l	31
6.1.3.64	cyl_hankel_2	31
6.1.3.65	cyl_hankel_2	31
6.1.3.66	cyl_hankel_2f	32
6.1.3.67	cyl_hankel_2f	32
6.1.3.68	cyl_hankel_2l	32
6.1.3.69	cyl_hankel_2l	32
6.1.3.70	dawson	32
6.1.3.71	dawsonf	33
6.1.3.72	dawsonl	33
6.1.3.73	digamma	33
6.1.3.74	digammaf	33
6.1.3.75	digammal	33
6.1.3.76	dilog	33
6.1.3.77	dilogf	33
6.1.3.78	dilogl	33
6.1.3.79	dirichlet_beta	34
6.1.3.80	dirichlet_betaf	34
6.1.3.81	dirichlet_betall	34
6.1.3.82	dirichlet_eta	34
6.1.3.83	dirichlet_etaf	34
6.1.3.84	dirichlet_etall	34

6.1.3.85	<code>double_factorial</code>	35
6.1.3.86	<code>double_factorialf</code>	35
6.1.3.87	<code>double_factoriall</code>	35
6.1.3.88	<code>ellint_cel</code>	35
6.1.3.89	<code>ellint_celf</code>	35
6.1.3.90	<code>ellint_cell</code>	35
6.1.3.91	<code>ellint_d</code>	35
6.1.3.92	<code>ellint_df</code>	36
6.1.3.93	<code>ellint_dl</code>	36
6.1.3.94	<code>ellint_el1</code>	36
6.1.3.95	<code>ellint_el1f</code>	36
6.1.3.96	<code>ellint_el1l</code>	36
6.1.3.97	<code>ellint_el2</code>	36
6.1.3.98	<code>ellint_el2f</code>	37
6.1.3.99	<code>ellint_el2l</code>	37
6.1.3.100	<code>ellint_el3</code>	37
6.1.3.101	<code>ellint_el3f</code>	37
6.1.3.102	<code>ellint_el3l</code>	37
6.1.3.103	<code>ellint_rc</code>	37
6.1.3.104	<code>ellint_rcf</code>	38
6.1.3.105	<code>ellint_rcl</code>	38
6.1.3.106	<code>ellint_rd</code>	38
6.1.3.107	<code>ellint_rdf</code>	39
6.1.3.108	<code>ellint_rdl</code>	39
6.1.3.109	<code>ellint_rf</code>	39
6.1.3.110	<code>ellint_rff</code>	39
6.1.3.111	<code>ellint_rfl</code>	40
6.1.3.112	<code>ellint_rg</code>	40
6.1.3.113	<code>ellint_rgf</code>	40
6.1.3.114	<code>ellint_rgl</code>	40
6.1.3.115	<code>ellint_rj</code>	41
6.1.3.116	<code>ellint_rjf</code>	41
6.1.3.117	<code>ellint_rjl</code>	41
6.1.3.118	<code>ellnome</code>	41
6.1.3.119	<code>ellnomef</code>	42
6.1.3.120	<code>ellnome1</code>	42
6.1.3.121	<code>expint_e1</code>	42

6.1.3.122 expint_e1f	42
6.1.3.123 expint_e1l	42
6.1.3.124 expint_en	42
6.1.3.125 expint_enf	42
6.1.3.126 expint_enl	42
6.1.3.127 factorial	42
6.1.3.128 factorialf	42
6.1.3.129 factoriall	43
6.1.3.130 fresnel_c	43
6.1.3.131 fresnel_cf	43
6.1.3.132 fresnel_cl	43
6.1.3.133 fresnel_s	43
6.1.3.134 fresnel_sf	43
6.1.3.135 fresnel_sl	43
6.1.3.136 gamma_l	43
6.1.3.137 gamma_lf	44
6.1.3.138 gamma_ll	44
6.1.3.139 gamma_p	44
6.1.3.140 gamma_pf	44
6.1.3.141 gamma_pl	44
6.1.3.142 gamma_q	44
6.1.3.143 gamma_qf	44
6.1.3.144 gamma_ql	44
6.1.3.145 gamma_u	44
6.1.3.146 gamma_uf	44
6.1.3.147 gamma_ul	44
6.1.3.148 gegenbauer	45
6.1.3.149 gegenbauerf	45
6.1.3.150 gegenbauerl	45
6.1.3.151 heuman_lambda	45
6.1.3.152 heuman_lambdaf	45
6.1.3.153 heuman_lambdal	45
6.1.3.154 hurwitz_zeta	45
6.1.3.155 hurwitz_zetaf	46
6.1.3.156 hurwitz_zetal	46
6.1.3.157 hyperg	46
6.1.3.158 hypergf	46

6.1.3.159 hypergl	46
6.1.3.160 ibeta	47
6.1.3.161 ibetac	47
6.1.3.162 ibetacf	47
6.1.3.163 ibetacl	47
6.1.3.164 ibetaf	47
6.1.3.165 ibetal	48
6.1.3.166 jacobi	48
6.1.3.167 jacobi_cn	48
6.1.3.168 jacobi_cnf	48
6.1.3.169 jacobi_cnl	48
6.1.3.170 jacobi_dn	49
6.1.3.171 jacobi_dnf	49
6.1.3.172 jacobi_dnl	49
6.1.3.173 jacobi_sn	49
6.1.3.174 jacobi_snf	49
6.1.3.175 jacobi_snl	50
6.1.3.176 jacobi_zeta	50
6.1.3.177 jacobi_zetaf	50
6.1.3.178 jacobi_zetal	50
6.1.3.179 jacobif	50
6.1.3.180 jacobil	50
6.1.3.181 lbincoef	51
6.1.3.182 lbincoeff	51
6.1.3.183 lbincoefl	51
6.1.3.184 ldouble_factorial	51
6.1.3.185 ldouble_factorialf	51
6.1.3.186 ldouble_factoriall	51
6.1.3.187 legendre_q	51
6.1.3.188 legendre_qf	51
6.1.3.189 legendre_ql	51
6.1.3.190 lfactorial	51
6.1.3.191 lfactorialf	51
6.1.3.192 lfactoriall	52
6.1.3.193 logint	52
6.1.3.194 logintf	52
6.1.3.195 logintl	52

6.1.3.196 lpochhammer_l	52
6.1.3.197 lpochhammer_lf	52
6.1.3.198 lpochhammer_ll	52
6.1.3.199 lpochhammer_u	53
6.1.3.200 lpochhammer_uf	53
6.1.3.201 lpochhammer_ul	53
6.1.3.202 owens_t	53
6.1.3.203 owens_tf	53
6.1.3.204 owens_tl	53
6.1.3.205 pochhammer_l	54
6.1.3.206 pochhammer_lf	54
6.1.3.207 pochhammer_ll	54
6.1.3.208 pochhammer_u	54
6.1.3.209 pochhammer_uf	54
6.1.3.210 pochhammer_ul	54
6.1.3.211 polylog	54
6.1.3.212 polylogf	54
6.1.3.213 polylogl	55
6.1.3.214 psi	55
6.1.3.215 psif	55
6.1.3.216 psil	55
6.1.3.217 radpoly	55
6.1.3.218 radpolyf	55
6.1.3.219 radpolyl	55
6.1.3.220 sinc	55
6.1.3.221 sinc_pi	56
6.1.3.222 sinc_pif	56
6.1.3.223 sinc_pil	56
6.1.3.224 sincf	56
6.1.3.225 sincl	56
6.1.3.226 sinhc	56
6.1.3.227 sinhc_pi	56
6.1.3.228 sinhc_pif	56
6.1.3.229 sinhc_pil	56
6.1.3.230 sinhcf	56
6.1.3.231 sinhcl	56
6.1.3.232 sinhlint	57

6.1.3.233 sinhinf	57
6.1.3.234 sinhintl	57
6.1.3.235 sinint	57
6.1.3.236 sinintf	57
6.1.3.237 sinintl	58
6.1.3.238 sph_bessel_i	58
6.1.3.239 sph_bessel_if	58
6.1.3.240 sph_bessel_il	58
6.1.3.241 sph_bessel_k	58
6.1.3.242 sph_bessel_kf	58
6.1.3.243 sph_bessel_kl	58
6.1.3.244 sph_hankel_1	58
6.1.3.245 sph_hankel_1	58
6.1.3.246 sph_hankel_1f	59
6.1.3.247 sph_hankel_1f	59
6.1.3.248 sph_hankel_1l	59
6.1.3.249 sph_hankel_1l	59
6.1.3.250 sph_hankel_2	59
6.1.3.251 sph_hankel_2	59
6.1.3.252 sph_hankel_2f	59
6.1.3.253 sph_hankel_2f	60
6.1.3.254 sph_hankel_2l	60
6.1.3.255 sph_hankel_2l	60
6.1.3.256 sph_harmonic	60
6.1.3.257 sph_harmonicf	60
6.1.3.258 sph_harmonicl	60
6.1.3.259 theta_1	61
6.1.3.260 theta_1f	61
6.1.3.261 theta_1l	61
6.1.3.262 theta_2	61
6.1.3.263 theta_2f	62
6.1.3.264 theta_2l	62
6.1.3.265 theta_3	62
6.1.3.266 theta_3f	62
6.1.3.267 theta_3l	63
6.1.3.268 theta_4	63
6.1.3.269 theta_4f	63

6.1.3.270	theta_4l	63
6.1.3.271	theta_c	63
6.1.3.272	theta_cf	64
6.1.3.273	theta_cl	64
6.1.3.274	theta_d	64
6.1.3.275	theta_df	64
6.1.3.276	theta_dl	65
6.1.3.277	theta_n	65
6.1.3.278	theta_nf	65
6.1.3.279	theta_nl	65
6.1.3.280	theta_s	65
6.1.3.281	theta_sf	66
6.1.3.282	theta_sl	66
6.1.3.283	zernike	66
6.1.3.284	zernikef	66
6.1.3.285	zernikel	66
6.2	Mathematical Special Functions	67
6.2.1	Detailed Description	68
6.2.2	Function Documentation	69
6.2.2.1	assoc_laguerre	69
6.2.2.2	assoc_laguerref	69
6.2.2.3	assoc_laguerrel	69
6.2.2.4	assoc_legendre	69
6.2.2.5	assoc_legendref	70
6.2.2.6	assoc_legendrel	70
6.2.2.7	beta	70
6.2.2.8	betaf	70
6.2.2.9	betal	71
6.2.2.10	comp_ellint_1	71
6.2.2.11	comp_ellint_1f	71
6.2.2.12	comp_ellint_1l	71
6.2.2.13	comp_ellint_2	72
6.2.2.14	comp_ellint_2f	72
6.2.2.15	comp_ellint_2l	72
6.2.2.16	comp_ellint_3	72
6.2.2.17	comp_ellint_3f	73
6.2.2.18	comp_ellint_3l	73

6.2.2.19	cyl_bessel_i	73
6.2.2.20	cyl_bessel_if	73
6.2.2.21	cyl_bessel_il	74
6.2.2.22	cyl_bessel_j	74
6.2.2.23	cyl_bessel_jf	74
6.2.2.24	cyl_bessel_jl	74
6.2.2.25	cyl_bessel_k	74
6.2.2.26	cyl_bessel_kf	75
6.2.2.27	cyl_bessel_kl	75
6.2.2.28	cyl_neumann	75
6.2.2.29	cyl_neumannf	76
6.2.2.30	cyl_neumannl	76
6.2.2.31	ellint_1	76
6.2.2.32	ellint_1f	76
6.2.2.33	ellint_1l	77
6.2.2.34	ellint_2	77
6.2.2.35	ellint_2f	77
6.2.2.36	ellint_2l	77
6.2.2.37	ellint_3	78
6.2.2.38	ellint_3f	78
6.2.2.39	ellint_3l	78
6.2.2.40	expint	78
6.2.2.41	expintf	79
6.2.2.42	expintl	79
6.2.2.43	hermite	79
6.2.2.44	hermitef	79
6.2.2.45	hermitel	80
6.2.2.46	laguerre	80
6.2.2.47	laguerref	80
6.2.2.48	laguerrel	80
6.2.2.49	legendre	80
6.2.2.50	legendref	81
6.2.2.51	legendrel	81
6.2.2.52	riemann_zeta	81
6.2.2.53	riemann_zetaf	81
6.2.2.54	riemann_zetal	82
6.2.2.55	sph_bessel	82

6.2.2.56	sph_besself	82
6.2.2.57	sph_bessell	82
6.2.2.58	sph_legendre	82
6.2.2.59	sph_legendref	83
6.2.2.60	sph_legendrel	83
6.2.2.61	sph_neumann	83
6.2.2.62	sph_neumannf	83
6.2.2.63	sph_neumannl	84
7	Namespace Documentation	85
7.1	__gnu_cxx Namespace Reference	85
7.2	std Namespace Reference	93
7.3	std::__detail Namespace Reference	95
7.3.1	Enumeration Type Documentation	112
7.3.1.1	anonymous enum	112
7.3.2	Function Documentation	112
7.3.2.1	__airy	112
7.3.2.2	__airy	113
7.3.2.3	__airy_ai	115
7.3.2.4	__airy_arg	115
7.3.2.5	__airy_asymp_absarg_ge_pio3	115
7.3.2.6	__airy_asymp_absarg_lt_pio3	116
7.3.2.7	__airy_bessel_i	116
7.3.2.8	__airy_bessel_k	117
7.3.2.9	__airy_bi	118
7.3.2.10	__airy_hyperg_rational	118
7.3.2.11	__assoc_laguerre	119
7.3.2.12	__assoc_legendre_p	119
7.3.2.13	__bernoulli	119
7.3.2.14	__bernoulli_2n	120
7.3.2.15	__bernoulli_series	120
7.3.2.16	__beta	120
7.3.2.17	__beta_gamma	121
7.3.2.18	__beta_inc	121
7.3.2.19	__beta_inc_cont_frac	122
7.3.2.20	__beta_lgamma	122
7.3.2.21	__beta_product	122

7.3.2.22	__bincoef	123
7.3.2.23	__binomial_cdf	123
7.3.2.24	__binomial_cdfc	123
7.3.2.25	__bose_einstein	124
7.3.2.26	__chebyshev_recur	124
7.3.2.27	__chebyshev_t	124
7.3.2.28	__chebyshev_u	124
7.3.2.29	__chebyshev_v	124
7.3.2.30	__chebyshev_w	125
7.3.2.31	__chi_squared_pdf	125
7.3.2.32	__chi_squared_pdfc	125
7.3.2.33	__chshint	125
7.3.2.34	__chshint_cont_frac	126
7.3.2.35	__chshint_series	126
7.3.2.36	__clamp_0_m2pi	126
7.3.2.37	__clamp_pi	126
7.3.2.38	__clausen	126
7.3.2.39	__clausen	126
7.3.2.40	__clausen_c	127
7.3.2.41	__clausen_c	127
7.3.2.42	__clausen_s	128
7.3.2.43	__clausen_s	128
7.3.2.44	__comp_ellint_1	128
7.3.2.45	__comp_ellint_2	129
7.3.2.46	__comp_ellint_3	129
7.3.2.47	__comp_ellint_d	130
7.3.2.48	__comp_ellint_rf	130
7.3.2.49	__comp_ellint_rg	130
7.3.2.50	__conf_hyperg	130
7.3.2.51	__conf_hyperg_lim	130
7.3.2.52	__conf_hyperg_lim_series	131
7.3.2.53	__conf_hyperg_luke	131
7.3.2.54	__conf_hyperg_series	131
7.3.2.55	__coshint	132
7.3.2.56	__cyl_bessel	132
7.3.2.57	__cyl_bessel_i	132
7.3.2.58	__cyl_bessel_ij_series	133

7.3.2.59	__cyl_bessel_ik	133
7.3.2.60	__cyl_bessel_ik_asymp	134
7.3.2.61	__cyl_bessel_ik_steel	134
7.3.2.62	__cyl_bessel_j	134
7.3.2.63	__cyl_bessel_jn	135
7.3.2.64	__cyl_bessel_jn_asymp	135
7.3.2.65	__cyl_bessel_jn_steel	135
7.3.2.66	__cyl_bessel_k	136
7.3.2.67	__cyl_hankel_1	136
7.3.2.68	__cyl_hankel_1	137
7.3.2.69	__cyl_hankel_2	137
7.3.2.70	__cyl_hankel_2	137
7.3.2.71	__cyl_neumann	138
7.3.2.72	__cyl_neumann_n	138
7.3.2.73	__dawson	139
7.3.2.74	__dawson_const_frac	139
7.3.2.75	__dawson_series	139
7.3.2.76	__debye_region	139
7.3.2.77	__dilog	139
7.3.2.78	__dirichlet_beta	140
7.3.2.79	__dirichlet_beta	140
7.3.2.80	__dirichlet_eta	140
7.3.2.81	__dirichlet_eta	141
7.3.2.82	__double_factorial	141
7.3.2.83	__ellint_1	141
7.3.2.84	__ellint_2	142
7.3.2.85	__ellint_3	142
7.3.2.86	__ellint_cel	143
7.3.2.87	__ellint_d	143
7.3.2.88	__ellint_el1	143
7.3.2.89	__ellint_el2	143
7.3.2.90	__ellint_el3	143
7.3.2.91	__ellint_rc	143
7.3.2.92	__ellint_rd	144
7.3.2.93	__ellint_rf	144
7.3.2.94	__ellint_rg	145
7.3.2.95	__ellint_rj	145

7.3.2.96	<code>__ellnome</code>	146
7.3.2.97	<code>__ellnome_k</code>	146
7.3.2.98	<code>__ellnome_series</code>	146
7.3.2.99	<code>__expint</code>	146
7.3.2.100	<code>__expint</code>	147
7.3.2.101	<code>__expint_asymp</code>	147
7.3.2.102	<code>__expint_E1</code>	148
7.3.2.103	<code>__expint_E1_asymp</code>	148
7.3.2.104	<code>__expint_E1_series</code>	148
7.3.2.105	<code>__expint_Ei</code>	149
7.3.2.106	<code>__expint_Ei_asymp</code>	149
7.3.2.107	<code>__expint_Ei_series</code>	150
7.3.2.108	<code>__expint_En_cont_frac</code>	150
7.3.2.109	<code>__expint_En_recursion</code>	150
7.3.2.110	<code>__expint_En_series</code>	151
7.3.2.111	<code>__expint_large_n</code>	151
7.3.2.112	<code>__f_cdf</code>	152
7.3.2.113	<code>__f_cdfc</code>	152
7.3.2.114	<code>__factorial</code>	152
7.3.2.115	<code>__fermi_dirac</code>	152
7.3.2.116	<code>__fock_airy</code>	153
7.3.2.117	<code>__fpequal</code>	153
7.3.2.118	<code>__fpimag</code>	154
7.3.2.119	<code>__fpimag</code>	155
7.3.2.120	<code>__fpreal</code>	155
7.3.2.121	<code>__fpreal</code>	155
7.3.2.122	<code>__fresnel</code>	155
7.3.2.123	<code>__fresnel_cont_frac</code>	156
7.3.2.124	<code>__fresnel_series</code>	156
7.3.2.125	<code>__gamma</code>	156
7.3.2.126	<code>__gamma_cont_frac</code>	156
7.3.2.127	<code>__gamma_l</code>	156
7.3.2.128	<code>__gamma_p</code>	157
7.3.2.129	<code>__gamma_q</code>	157
7.3.2.130	<code>__gamma_series</code>	157
7.3.2.131	<code>__gamma_temme</code>	157
7.3.2.132	<code>__gamma_u</code>	158

7.3.2.133	__gauss	158
7.3.2.134	__gegenbauer_poly	158
7.3.2.135	__hankel	158
7.3.2.136	__hankel_debye	159
7.3.2.137	__hankel_params	159
7.3.2.138	__hankel_uniform	159
7.3.2.139	__hankel_uniform_olver	160
7.3.2.140	__hankel_uniform_outer	160
7.3.2.141	__hankel_uniform_sum	161
7.3.2.142	__heuman_lambda	161
7.3.2.143	__hurwitz_zeta	161
7.3.2.144	__hurwitz_zeta_euler_maclaurin	162
7.3.2.145	__hydrogen	162
7.3.2.146	__hyperg	162
7.3.2.147	__hyperg_luke	163
7.3.2.148	__hyperg_reflect	163
7.3.2.149	__hyperg_series	163
7.3.2.150	__jacobi_sncndn	164
7.3.2.151	__jacobi_zeta	164
7.3.2.152	__laguerre	164
7.3.2.153	__log_bincoef	164
7.3.2.154	__log_double_factorial	165
7.3.2.155	__log_double_factorial	165
7.3.2.156	__log_factorial	165
7.3.2.157	__log_gamma	165
7.3.2.158	__log_gamma_bernoulli	166
7.3.2.159	__log_gamma_lanczos	166
7.3.2.160	__log_gamma_sign	166
7.3.2.161	__log_gamma_spouge	167
7.3.2.162	__log_pochhammer_l	167
7.3.2.163	__log_pochhammer_u	168
7.3.2.164	__logint	168
7.3.2.165	__owens_t	168
7.3.2.166	__pochhammer_l	169
7.3.2.167	__pochhammer_u	169
7.3.2.168	__poly_hermite	169
7.3.2.169	__poly_hermite_asymp	170

7.3.2.170	<code>__poly_hermite_recursion</code>	170
7.3.2.171	<code>__poly_jacobi</code>	171
7.3.2.172	<code>__poly_laguerre</code>	171
7.3.2.173	<code>__poly_laguerre_hyperg</code>	171
7.3.2.174	<code>__poly_laguerre_large_n</code>	173
7.3.2.175	<code>__poly_laguerre_recursion</code>	173
7.3.2.176	<code>__poly_legendre_p</code>	174
7.3.2.177	<code>__poly_legendre_q</code>	174
7.3.2.178	<code>__poly_radial_jacobi</code>	174
7.3.2.179	<code>__polylog</code>	175
7.3.2.180	<code>__polylog</code>	176
7.3.2.181	<code>__polylog_exp</code>	176
7.3.2.182	<code>__polylog_exp_asymp</code>	177
7.3.2.183	<code>__polylog_exp_int_neg</code>	177
7.3.2.184	<code>__polylog_exp_int_neg</code>	177
7.3.2.185	<code>__polylog_exp_int_pos</code>	178
7.3.2.186	<code>__polylog_exp_int_pos</code>	178
7.3.2.187	<code>__polylog_exp_neg</code>	179
7.3.2.188	<code>__polylog_exp_neg</code>	179
7.3.2.189	<code>__polylog_exp_neg_even</code>	179
7.3.2.190	<code>__polylog_exp_neg_odd</code>	180
7.3.2.191	<code>__polylog_exp_negative_real_part</code>	181
7.3.2.192	<code>__polylog_exp_pos</code>	181
7.3.2.193	<code>__polylog_exp_pos</code>	182
7.3.2.194	<code>__polylog_exp_pos</code>	182
7.3.2.195	<code>__polylog_exp_real_neg</code>	183
7.3.2.196	<code>__polylog_exp_real_neg</code>	183
7.3.2.197	<code>__polylog_exp_real_pos</code>	183
7.3.2.198	<code>__polylog_exp_real_pos</code>	184
7.3.2.199	<code>__psi</code>	184
7.3.2.200	<code>__psi</code>	184
7.3.2.201	<code>__psi_asymp</code>	185
7.3.2.202	<code>__psi_series</code>	185
7.3.2.203	<code>__riemann_zeta</code>	185
7.3.2.204	<code>__riemann_zeta_alt</code>	186
7.3.2.205	<code>__riemann_zeta_euler_maclaurin</code>	186
7.3.2.206	<code>__riemann_zeta_glob</code>	186

7.3.2.207	__riemann_zeta_m_1	187
7.3.2.208	__riemann_zeta_m_1_sum	188
7.3.2.209	__riemann_zeta_product	188
7.3.2.210	__riemann_zeta_sum	188
7.3.2.211	__sinc	189
7.3.2.212	__sinc	189
7.3.2.213	__sinc_pi	189
7.3.2.214	__sincosint	189
7.3.2.215	__sincosint_asymp	190
7.3.2.216	__sincosint_cont_frac	190
7.3.2.217	__sincosint_series	190
7.3.2.218	__sinhc	190
7.3.2.219	__sinhc	190
7.3.2.220	__sinhc_pi	190
7.3.2.221	__sinhint	191
7.3.2.222	__sph_bessel	191
7.3.2.223	__sph_bessel	191
7.3.2.224	__sph_bessel_ik	192
7.3.2.225	__sph_bessel_jn	193
7.3.2.226	__sph_hankel	193
7.3.2.227	__sph_hankel_1	194
7.3.2.228	__sph_hankel_1	194
7.3.2.229	__sph_hankel_2	194
7.3.2.230	__sph_hankel_2	195
7.3.2.231	__sph_harmonic	195
7.3.2.232	__sph_legendre	195
7.3.2.233	__sph_neumann	196
7.3.2.234	__sph_neumann	196
7.3.2.235	__students_t_cdf	197
7.3.2.236	__students_t_cdfc	197
7.3.2.237	__theta_1	197
7.3.2.238	__theta_2	198
7.3.2.239	__theta_2_asymp	198
7.3.2.240	__theta_2_sum	198
7.3.2.241	__theta_3	198
7.3.2.242	__theta_3_asymp	199
7.3.2.243	__theta_3_sum	199

7.3.2.244	__theta_4	199
7.3.2.245	__theta_c	199
7.3.2.246	__theta_d	199
7.3.2.247	__theta_n	199
7.3.2.248	__theta_s	200
7.3.2.249	__zernike	200
7.3.2.250	__znorm1	200
7.3.2.251	__znorm2	200
7.3.2.252	evenzeta	200
7.3.3	Variable Documentation	200
7.3.3.1	_Num_Euler_Maclaurin_zeta	200
7.3.3.2	_S_double_factorial_table	201
7.3.3.3	_S_Euler_Maclaurin_zeta	201
7.3.3.4	_S_factorial_table	201
7.3.3.5	_S_neg_double_factorial_table	201
7.3.3.6	_S_num_double_factorials	201
7.3.3.7	_S_num_double_factorials< double >	201
7.3.3.8	_S_num_double_factorials< float >	201
7.3.3.9	_S_num_double_factorials< long double >	201
7.3.3.10	_S_num_factorials	201
7.3.3.11	_S_num_factorials< double >	202
7.3.3.12	_S_num_factorials< float >	202
7.3.3.13	_S_num_factorials< long double >	202
7.3.3.14	_S_num_neg_double_factorials	202
7.3.3.15	_S_num_neg_double_factorials< double >	202
7.3.3.16	_S_num_neg_double_factorials< float >	202
7.3.3.17	_S_num_neg_double_factorials< long double >	202
7.3.3.18	_S_num_zetam1	202
7.3.3.19	_S_zetam1	202
8	Class Documentation	203
8.1	std::__detail::_Factorial_table< _Tp > Struct Template Reference	203
8.1.1	Detailed Description	203
8.1.2	Member Data Documentation	203
8.1.2.1	__factorial	203
8.1.2.2	__log_factorial	203
8.1.2.3	__n	203

9	File Documentation	205
9.1	bits/sf_airy.tcc File Reference	205
9.1.1	Detailed Description	207
9.1.2	Macro Definition Documentation	207
9.1.2.1	_GLIBCXX_BITS_SF_AIRY_TCC	207
9.2	bits/sf_bessel.tcc File Reference	207
9.2.1	Detailed Description	209
9.2.2	Macro Definition Documentation	209
9.2.2.1	_GLIBCXX_BITS_SF_BESSEL_TCC	209
9.3	bits/sf_beta.tcc File Reference	209
9.3.1	Detailed Description	211
9.3.2	Macro Definition Documentation	211
9.3.2.1	_GLIBCXX_BITS_SF_BETA_TCC	211
9.4	bits/sf_cardinal.tcc File Reference	211
9.4.1	Macro Definition Documentation	213
9.4.1.1	_GLIBCXX_BITS_SF_CARDINAL_TCC	213
9.5	bits/sf_chebyshev.tcc File Reference	213
9.5.1	Detailed Description	214
9.5.2	Macro Definition Documentation	214
9.5.2.1	_GLIBCXX_SF_CHEBYSHEV_TCC	214
9.6	bits/sf_dawson.tcc File Reference	215
9.6.1	Detailed Description	216
9.6.2	Macro Definition Documentation	216
9.6.2.1	_GLIBCXX_SF_DAWSON_TCC	216
9.7	bits/sf_ellint.tcc File Reference	216
9.7.1	Detailed Description	218
9.7.2	Macro Definition Documentation	218
9.7.2.1	_GLIBCXX_BITS_SF_ELLINT_TCC	218
9.8	bits/sf_expint.tcc File Reference	219
9.8.1	Detailed Description	221
9.8.2	Macro Definition Documentation	221
9.8.2.1	_GLIBCXX_BITS_SF_EXPINT_TCC	221
9.9	bits/sf_fresnel.tcc File Reference	221
9.9.1	Detailed Description	222
9.9.2	Macro Definition Documentation	222
9.9.2.1	_GLIBCXX_SF_FRESNEL_TCC	222
9.10	bits/sf_gamma.tcc File Reference	222

9.10.1 Detailed Description	227
9.10.2 Macro Definition Documentation	228
9.10.2.1 _GLIBCXX_BITS_SF_GAMMA_TCC	228
9.11 bits/sf_gegenbauer.tcc File Reference	228
9.11.1 Detailed Description	229
9.11.2 Macro Definition Documentation	229
9.11.2.1 _GLIBCXX_SF_GEGENBAUER_TCC	229
9.12 bits/sf_hankel.tcc File Reference	229
9.12.1 Detailed Description	232
9.12.2 Macro Definition Documentation	232
9.12.2.1 _GLIBCXX_BITS_SF_HANKEL_TCC	232
9.13 bits/sf_hankel_new.tcc File Reference	232
9.13.1 Macro Definition Documentation	232
9.13.1.1 _GLIBCXX_BITS_SF_HANKEL_NEW_TCC	232
9.14 bits/sf_hermite.tcc File Reference	233
9.14.1 Detailed Description	234
9.14.2 Macro Definition Documentation	234
9.14.2.1 _GLIBCXX_BITS_SF_HERMITE_TCC	234
9.15 bits/sf_hydrogen.tcc File Reference	234
9.15.1 Detailed Description	235
9.15.2 Macro Definition Documentation	235
9.15.2.1 _GLIBCXX_BITS_SF_HYDROGEN_TCC	235
9.16 bits/sf_hyperg.tcc File Reference	235
9.16.1 Detailed Description	237
9.16.2 Macro Definition Documentation	237
9.16.2.1 _GLIBCXX_BITS_SF_HYPERG_TCC	237
9.17 bits/sf_hypint.tcc File Reference	237
9.17.1 Detailed Description	238
9.17.2 Macro Definition Documentation	238
9.17.2.1 _GLIBCXX_SF_HYPINT_TCC	238
9.18 bits/sf_jacobi.tcc File Reference	239
9.18.1 Detailed Description	240
9.18.2 Macro Definition Documentation	240
9.18.2.1 _GLIBCXX_SF_JACOBI_TCC	240
9.19 bits/sf_laguerre.tcc File Reference	240
9.19.1 Detailed Description	242
9.19.2 Macro Definition Documentation	242

9.19.2.1	_GLIBCXX_BITS_SF_LAGUERRE_TCC	242
9.20	bits/sf_legendre.tcc File Reference	242
9.20.1	Detailed Description	243
9.20.2	Macro Definition Documentation	243
9.20.2.1	_GLIBCXX_BITS_SF_LEGENDRE_TCC	243
9.21	bits/sf_mod_bessel.tcc File Reference	243
9.21.1	Detailed Description	245
9.21.2	Macro Definition Documentation	245
9.21.2.1	_GLIBCXX_BITS_SF_MOD_BESSEL_TCC	245
9.22	bits/sf_owens_t.tcc File Reference	246
9.22.1	Detailed Description	246
9.22.2	Macro Definition Documentation	246
9.22.2.1	_GLIBCXX_BITS_SF_OWENS_T_TCC	246
9.23	bits/sf_polylog.tcc File Reference	247
9.23.1	Detailed Description	249
9.23.2	Macro Definition Documentation	249
9.23.2.1	_GLIBCXX_BITS_SF_POLYLOG_TCC	249
9.24	bits/sf_theta.tcc File Reference	249
9.24.1	Detailed Description	251
9.24.2	Macro Definition Documentation	251
9.24.2.1	_GLIBCXX_SF_THETA_TCC	251
9.25	bits/sf_trigint.tcc File Reference	252
9.25.1	Detailed Description	253
9.25.2	Macro Definition Documentation	253
9.25.2.1	_GLIBCXX_SF_TRIGINT_TCC	253
9.26	bits/sf_zeta.tcc File Reference	253
9.26.1	Detailed Description	255
9.26.2	Macro Definition Documentation	255
9.26.2.1	_GLIBCXX_BITS_SF_ZETA_TCC	255
9.27	bits/specfun.h File Reference	256
9.27.1	Detailed Description	266
9.27.2	Macro Definition Documentation	266
9.27.2.1	__cpp_lib_math_special_functions	266
9.27.2.2	__STDCPP_MATH_SPEC_FUNCS__	266

Chapter 1

Todo List

Member `std::__detail::__dawson_const_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.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Extended Mathematical Special Functions	11
Mathematical Special Functions	67

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

__gnu_cxx	85
std	93
std::__detail	95

Chapter 4

Class Index

4.1 Class List

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

[std::__detail::_Factorial_table<_Tp>](#) 203

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

bits/sf_airy.tcc	205
bits/sf_bessel.tcc	207
bits/sf_beta.tcc	209
bits/sf_cardinal.tcc	211
bits/sf_chebyshev.tcc	213
bits/sf_dawson.tcc	215
bits/sf_ellint.tcc	216
bits/sf_expint.tcc	219
bits/sf_fresnel.tcc	221
bits/sf_gamma.tcc	222
bits/sf_gegenbauer.tcc	228
bits/sf_hankel.tcc	229
bits/sf_hankel_new.tcc	232
bits/sf_hermite.tcc	233
bits/sf_hydrogen.tcc	234
bits/sf_hyperg.tcc	235
bits/sf_hypint.tcc	237
bits/sf_jacobi.tcc	239
bits/sf_laguerre.tcc	240
bits/sf_legendre.tcc	242
bits/sf_mod_bessel.tcc	243
bits/sf_owens_t.tcc	246
bits/sf_polylog.tcc	247
bits/sf_theta.tcc	249
bits/sf_trigint.tcc	252
bits/sf_zeta.tcc	253
bits/specfun.h	256

Chapter 6

Module Documentation

6.1 Extended Mathematical Special Functions

Enumerations

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

Functions

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

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

- `template<typename _Tpnu, typename _Tp>`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (_Tpnu __nu, _Tp`
`__z)`
- `template<typename _Tpnu, typename _Tp>`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_1 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp>`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp`
`__z)`
- `template<typename _Tpnu, typename _Tp>`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (std::complex< _Tpnu > __nu, std::complex< _Tp > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tp>`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::dawson (_Tp __x)`
- `float __gnu_cxx::dawsonf (float __x)`
- `long double __gnu_cxx::dawsonl (long double __x)`
- `template<typename _Tp>`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::digamma (_Tp __z)`
- `float __gnu_cxx::digammaf (float __z)`
- `long double __gnu_cxx::digammal (long double __z)`
- `template<typename _Tp>`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::dilog (_Tp __x)`
- `float __gnu_cxx::dilogf (float __x)`
- `long double __gnu_cxx::dilogl (long double __x)`
- `template<typename _Tp>`
`_Tp __gnu_cxx::dirichlet_beta (_Tp __x)`
- `float __gnu_cxx::dirichlet_betaf (float __x)`
- `long double __gnu_cxx::dirichlet_betall (long double __x)`
- `template<typename _Tp>`
`_Tp __gnu_cxx::dirichlet_eta (_Tp __x)`
- `float __gnu_cxx::dirichlet_etaf (float __x)`
- `long double __gnu_cxx::dirichlet_etaall (long double __x)`
- `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_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_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)
- float [__gnu_cxx::hurwitz_zetaf](#) (float __s, float __a)
- long double [__gnu_cxx::hurwitz_zetal](#) (long double __s, long double __a)
- template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >
 [__gnu_cxx::__promote_4<_Tpa, _Tpb, _Tpc, _Tp>::__type __gnu_cxx::hyperg](#) (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)
- float [__gnu_cxx::hypergf](#) (float __a, float __b, float __c, float __x)
- long double [__gnu_cxx::hypergl](#) (long double __a, long double __b, long double __c, long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
 [__gnu_cxx::__promote_num_t<_Ta, _Tb, _Tp> __gnu_cxx::ibeta](#) (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta, typename _Tb, typename _Tp >
 [__gnu_cxx::__promote_num_t<_Ta, _Tb, _Tp> __gnu_cxx::ibetac](#) (_Ta __a, _Tb __b, _Tp __x)
- float [__gnu_cxx::ibetacf](#) (float __a, float __b, float __x)
- long double [__gnu_cxx::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::jacobi (unsigned __n, float __alpha, float __beta, float __x)`
- `long double __gnu_cxx::jacobil (unsigned __n, long double __alpha, long double __beta, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_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 >
 std::complex< [__gnu_cxx::__promote_num_t<_Tp > __gnu_cxx::polylog](#) (_Tp __s, std::complex< _Tp > __w)
- std::complex< float > [__gnu_cxx::polylogf](#) (float __s, std::complex< float > __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`)

6.1.1 Detailed Description

A collection of advanced mathematical special functions.

6.1.2 Enumeration Type Documentation

6.1.2.1 anonymous enum

Enumerator

`__GLIBCXX_JACOBI_SN`
`__GLIBCXX_JACOBI_CN`
`__GLIBCXX_JACOBI_DN`

Definition at line 1433 of file `specfun.h`.

6.1.3 Function Documentation

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

Definition at line 2047 of file `specfun.h`.

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

Definition at line 2027 of file `specfun.h`.

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

Definition at line 2035 of file `specfun.h`.

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

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

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

Definition at line 2058 of file specfun.h.

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

Definition at line 2066 of file specfun.h.

6.1.3.7 `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 2963 of file specfun.h.

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

Definition at line 2944 of file specfun.h.

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

Definition at line 2948 of file specfun.h.

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

Definition at line 2914 of file specfun.h.

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

Definition at line 2902 of file specfun.h.

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

Definition at line 2906 of file specfun.h.

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

Return the Chebyshev polynomials of the first kind of order n and argument x .

The Chebyshev polynomials of the first kind is defined by

Parameters

<code>__n</code>	
<code>__x</code>	

Definition at line 1614 of file specfun.h.

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

Return the Chebyshev polynomials of the first kind of order `n` and argument `x`.

See also

[chebyshev_t](#) for details.

Definition at line 1587 of file specfun.h.

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

Return the Chebyshev polynomials of the first kind of order `n` and argument `x`.

See also

[chebyshev_t](#) for details.

Definition at line 1597 of file specfun.h.

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

Return the Chebyshev polynomials of the second kind of order `n` and argument `x`.

The Chebyshev polynomials of the second kind is defined by

Parameters

<code>__n</code>	
<code>__x</code>	

Definition at line 1656 of file specfun.h.

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

Return the Chebyshev polynomials of the second kind of order `n` and argument `x`.

See also

[chebyshev_u](#) for details.

Definition at line 1629 of file specfun.h.

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

Return the Chebyshev polynomials of the second kind of order `n` and argument `x`.

See also

[chebyshev_u](#) for details.

Definition at line 1639 of file `specfun.h`.

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

Return the Chebyshev polynomials of the third kind of order `n` and argument `x`.

The Chebyshev polynomials of the third kind is defined by

Parameters

<code>__n</code>	
<code>__x</code>	

Definition at line 1698 of file `specfun.h`.

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

Return the Chebyshev polynomials of the third kind of order `n` and argument `x`.

See also

[chebyshev_v](#) for details.

Definition at line 1671 of file `specfun.h`.

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

Return the Chebyshev polynomials of the third kind of order `n` and argument `x`.

See also

[chebyshev_v](#) for details.

Definition at line 1681 of file `specfun.h`.

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

Return the Chebyshev polynomials of the fourth kind of order `n` and argument `x`.

The Chebyshev polynomials of the fourth kind is defined by

Parameters

<code>__n</code>	
<code>__x</code>	

Definition at line 1740 of file specfun.h.

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

Return the Chebyshev polynomials of the fourth kind of order `n` and argument `x`.

See also

[chebyshev_w](#) for details.

Definition at line 1713 of file specfun.h.

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

Return the Chebyshev polynomials of the fourth kind of order `n` and argument `x`.

See also

[chebyshev_w](#) for details.

Definition at line 1723 of file specfun.h.

6.1.3.25 `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 3683 of file specfun.h.

6.1.3.26 `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 3704 of file specfun.h.

6.1.3.27 `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 3644 of file specfun.h.

6.1.3.28 `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 3619 of file specfun.h.

6.1.3.29 `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 3628 of file specfun.h.

6.1.3.30 `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 3605 of file specfun.h.

6.1.3.31 `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 3580 of file specfun.h.

6.1.3.32 `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 3589 of file `specfun.h`.

6.1.3.33 `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 3658 of file `specfun.h`.

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

Definition at line 3692 of file `specfun.h`.

6.1.3.35 `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 3667 of file `specfun.h`.

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

Definition at line 3696 of file `specfun.h`.

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

Return the complete Legendre elliptic integral D of k and ϕ .

The complete Legendre elliptic integral D is defined by

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

Parameters

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

Definition at line 3112 of file specfun.h.

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

Definition at line 3093 of file specfun.h.

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

Definition at line 3097 of file specfun.h.

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

Definition at line 2173 of file specfun.h.

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

Definition at line 2177 of file specfun.h.

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

Definition at line 2185 of file specfun.h.

6.1.3.43 `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 2388 of file specfun.h.

6.1.3.44 `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 2397 of file specfun.h.

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

Definition at line 2405 of file specfun.h.

6.1.3.46 `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 1106 of file specfun.h.

6.1.3.47 `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 1202 of file specfun.h.

6.1.3.48 `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 1173 of file specfun.h.

6.1.3.49 `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 1183 of file `specfun.h`.

6.1.3.50 `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 1074 of file `specfun.h`.

6.1.3.51 `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 1085 of file `specfun.h`.

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

Return the hyperbolic cosine integral of argument `x`.

The hyperbolic cosine integral is defined by

Parameters

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

Definition at line 1426 of file `specfun.h`.

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

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

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

Return the hyperbolic cosine integral of argument x .

See also

[coshint](#) for details.

Definition at line 1411 of file specfun.h.

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

Return the cosine integral of argument x .

The cosine integral is defined by

Parameters

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

Definition at line 1355 of file specfun.h.

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

Return the cosine integral of argument x .

See also

[cosint](#) for details.

Definition at line 1331 of file specfun.h.

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

Return the cosine integral of argument x .

See also

[cosint](#) for details.

Definition at line 1340 of file specfun.h.

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

Definition at line 1887 of file specfun.h.

6.1.3.59 `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 of complex order ν and complex argument 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 complex order
<code>__x</code>	The complex argument

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

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

Definition at line 1875 of file `specfun.h`.

6.1.3.61 `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 of complex order ν and complex argument x .

See also

[cyl_hankel_1](#) for more details.

Definition at line 3279 of file `specfun.h`.

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

Definition at line 1879 of file `specfun.h`.

6.1.3.63 `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 of complex order ν and complex argument x .

See also

[cyl_hankel_1](#) for more details.

Definition at line 3289 of file `specfun.h`.

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

Definition at line 1908 of file `specfun.h`.

6.1.3.65 `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 of complex order ν and complex argument 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 complex order
<code>__x</code>	The complex argument

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

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

Definition at line 1896 of file `specfun.h`.

6.1.3.67 `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 of complex order ν and complex argument x .

See also

[cyl_hankel_2](#) for more details.

Definition at line 3321 of file `specfun.h`.

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

Definition at line 1900 of file `specfun.h`.

6.1.3.69 `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 of complex order ν and complex argument x .

See also

[cyl_hankel_2](#) for more details.

Definition at line 3331 of file `specfun.h`.

6.1.3.70 `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 $-\inf < x < \inf$.
------------------	-----------------------------------

Definition at line 2690 of file `specfun.h`.

6.1.3.71 `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 2662 of file `specfun.h`.

6.1.3.72 `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 2671 of file `specfun.h`.

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

Definition at line 2143 of file `specfun.h`.

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

Definition at line 2131 of file `specfun.h`.

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

Definition at line 2135 of file `specfun.h`.

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

Definition at line 2164 of file `specfun.h`.

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

Definition at line 2152 of file `specfun.h`.

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

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

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

Return the Dirichlet beta function of real argument x .

The Dirichlet beta function is defined by

Parameters

<code>__x</code>	
------------------	--

Definition at line 3566 of file `specfun.h`.

6.1.3.80 `float __gnu_cxx::dirichlet_betaf (float __x) [inline]`

Definition at line 3547 of file `specfun.h`.

6.1.3.81 `long double __gnu_cxx::dirichlet_betad (long double __x) [inline]`

Definition at line 3551 of file `specfun.h`.

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

Return the Dirichlet eta function of real argument x .

The Dirichlet eta function is defined by

Parameters

<code>__x</code>	
------------------	--

Definition at line 3538 of file `specfun.h`.

6.1.3.83 `float __gnu_cxx::dirichlet_etaf (float __x) [inline]`

Return the Dirichlet eta function of real argument x .

See also

[dirichlet_eta](#) for details.

Definition at line 3514 of file `specfun.h`.

6.1.3.84 `long double __gnu_cxx::dirichlet_etad (long double __x) [inline]`

Return the Dirichlet eta function of real argument x .

See also

[dirichlet_eta](#) for details.

Definition at line 3523 of file `specfun.h`.

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

Definition at line 2851 of file specfun.h.

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

Definition at line 2839 of file specfun.h.

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

Definition at line 2843 of file specfun.h.

6.1.3.88 `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 of ...

The Bulirsch complete elliptic integral is defined by

Parameters

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

Definition at line 3264 of file specfun.h.

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

Definition at line 3241 of file specfun.h.

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

Definition at line 3245 of file specfun.h.

6.1.3.91 `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 of k and 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 $-1 \leq k \leq +1$
<code>__phi</code>	The angle

Definition at line 3141 of file specfun.h.

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

Definition at line 3121 of file specfun.h.

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

Definition at line 3125 of file specfun.h.

6.1.3.94 `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 of the first kind of ...

The Bulirsch elliptic integral of the first kind is defined by

Parameters

<code>__x</code>	The argument
<code>__k_c</code>	The complementary modulus $k_c = \sqrt{1 - k^2}$

Definition at line 3170 of file specfun.h.

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

Definition at line 3150 of file specfun.h.

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

Definition at line 3154 of file specfun.h.

6.1.3.97 `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 of ...

The Bulirsch elliptic integral of the second kind is defined by

Parameters

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

Definition at line 3202 of file specfun.h.

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

Definition at line 3179 of file specfun.h.

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

Definition at line 3183 of file specfun.h.

6.1.3.100 `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 of ...

The Bulirsch elliptic integral of the third kind is defined by

Parameters

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

Definition at line 3232 of file specfun.h.

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

Definition at line 3211 of file specfun.h.

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

Definition at line 3215 of file specfun.h.

6.1.3.103 `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 2275 of file specfun.h.

6.1.3.104 `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 2241 of file specfun.h.

6.1.3.105 `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 2250 of file specfun.h.

6.1.3.106 `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 2374 of file specfun.h.

6.1.3.107 `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 2338 of file specfun.h.

6.1.3.108 `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 2347 of file specfun.h.

6.1.3.109 `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.

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 2227 of file specfun.h.

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

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

See also

[ellint_rf](#) for details.

Definition at line 2199 of file specfun.h.

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

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

See also

[ellint_rf](#) for details.

Definition at line 2208 of file specfun.h.

6.1.3.112 `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 2454 of file specfun.h.

6.1.3.113 `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 2419 of file specfun.h.

6.1.3.114 `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 2428 of file specfun.h.

6.1.3.115 `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 2324 of file `specfun.h`.

6.1.3.116 `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 2289 of file `specfun.h`.

6.1.3.117 `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 2298 of file `specfun.h`.

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

Return the elliptic nome function of modulus k .

The elliptic nome function is defined by

Parameters

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

Definition at line 3892 of file specfun.h.

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

Definition at line 3873 of file specfun.h.

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

Definition at line 3877 of file specfun.h.

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

Definition at line 2706 of file specfun.h.

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

Definition at line 2697 of file specfun.h.

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

Definition at line 2701 of file specfun.h.

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

Definition at line 2725 of file specfun.h.

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

Definition at line 2713 of file specfun.h.

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

Definition at line 2717 of file specfun.h.

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

Definition at line 2830 of file specfun.h.

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

Definition at line 2818 of file specfun.h.

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

Definition at line 2822 of file specfun.h.

6.1.3.130 `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 2648 of file specfun.h.

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

Definition at line 2629 of file specfun.h.

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

Definition at line 2633 of file specfun.h.

6.1.3.133 `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 2620 of file specfun.h.

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

Definition at line 2601 of file specfun.h.

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

Definition at line 2605 of file specfun.h.

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

Definition at line 2122 of file specfun.h.

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

Definition at line 2110 of file specfun.h.

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

Definition at line 2114 of file specfun.h.

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

Definition at line 3005 of file specfun.h.

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

Definition at line 2993 of file specfun.h.

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

Definition at line 2997 of file specfun.h.

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

Definition at line 3026 of file specfun.h.

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

Definition at line 3014 of file specfun.h.

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

Definition at line 3018 of file specfun.h.

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

Definition at line 2101 of file specfun.h.

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

Definition at line 2089 of file specfun.h.

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

Definition at line 2093 of file specfun.h.

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

Definition at line 1782 of file specfun.h.

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

Definition at line 1770 of file specfun.h.

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

Definition at line 1774 of file specfun.h.

6.1.3.151 `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 of k and $@c\phi$.

The complete Heuman lambda function is defined by

Parameters

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

Definition at line 3084 of file specfun.h.

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

Definition at line 3064 of file specfun.h.

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

Definition at line 3068 of file specfun.h.

6.1.3.154 `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 argument s , and parameter a .

The the Hurwitz zeta function is defined by

$$\zeta(s, a) =$$

Parameters

<code>__s</code>	The argument
------------------	--------------

<code>__a</code>	The parameter
------------------	---------------

Definition at line 2483 of file specfun.h.

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

Definition at line 2463 of file specfun.h.

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

Definition at line 2467 of file specfun.h.

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

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

The hypergeometric function is defined by

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

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 1155 of file specfun.h.

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

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

See also

[hyperg](#) for details.

Definition at line 1122 of file specfun.h.

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

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

See also

[hyperg](#) for details.

Definition at line 1133 of file specfun.h.

6.1.3.160 `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 2561 of file `specfun.h`.

6.1.3.161 `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 2592 of file `specfun.h`.

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

Definition at line 2570 of file `specfun.h`.

References `__gnu_cxx::ibetaf()`.

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

Definition at line 2574 of file `specfun.h`.

References `__gnu_cxx::ibetac()`.

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

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

See `ibeta` for details.

Definition at line 2526 of file `specfun.h`.

Referenced by `__gnu_cxx::ibetaf()`.

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

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

See `ibeta` for details.

Definition at line 2536 of file `specfun.h`.

Referenced by `__gnu_cxx::ibetacl()`.

6.1.3.166 `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]`

Definition at line 1761 of file `specfun.h`.

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

6.1.3.167 `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` integral of modulus `k` and argument `u`.

The Jacobi elliptic `cn` integral is defined by

Parameters

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

Definition at line 1525 of file `specfun.h`.

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

Return the Jacobi elliptic `cn` integral of modulus `k` and argument `u`.

See also

[jacobi_cn](#) for details.

Definition at line 1494 of file `specfun.h`.

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

Return the Jacobi elliptic `cn` integral of modulus `k` and argument `u`.

See also

[jacobi_cn](#) for details.

Definition at line 1506 of file `specfun.h`.

6.1.3.170 `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` integral of modulus `k` and argument `u`.

The Jacobi elliptic `dn` integral is defined by

Parameters

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

Definition at line 1571 of file `specfun.h`.

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

Return the Jacobi elliptic `dn` integral of modulus `k` and argument `u`.

See also

[jacobi_dn](#) for details.

Definition at line 1540 of file `specfun.h`.

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

Return the Jacobi elliptic `dn` integral of modulus `k` and argument `u`.

See also

[jacobi_dn](#) for details.

Definition at line 1552 of file `specfun.h`.

6.1.3.173 `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` integral of modulus `k` and argument `u`.

The Jacobi elliptic `sn` integral is defined by

Parameters

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

Definition at line 1479 of file `specfun.h`.

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

Return the Jacobi elliptic `sn` integral of modulus `k` and argument `u`.

See also

[jacobi_sn](#) for details.

Definition at line 1448 of file specfun.h.

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

Return the Jacobi elliptic `sn` integral of modulus `k` and argument `u`.

See also

[jacobi_sn](#) for details.

Definition at line 1460 of file specfun.h.

6.1.3.176 `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

Parameters

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

Definition at line 3055 of file specfun.h.

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

Definition at line 3035 of file specfun.h.

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

Definition at line 3039 of file specfun.h.

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

Definition at line 1749 of file specfun.h.

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

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

Definition at line 1753 of file specfun.h.

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

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

Definition at line 2935 of file specfun.h.

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

Definition at line 2923 of file specfun.h.

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

Definition at line 2927 of file specfun.h.

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

Definition at line 2893 of file specfun.h.

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

Definition at line 2881 of file specfun.h.

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

Definition at line 2885 of file specfun.h.

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

Definition at line 2984 of file specfun.h.

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

Definition at line 2972 of file specfun.h.

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

Definition at line 2976 of file specfun.h.

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

Definition at line 2872 of file specfun.h.

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

Definition at line 2860 of file specfun.h.

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

Definition at line 2864 of file specfun.h.

6.1.3.193 `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

Parameters

<code>__x</code>	
------------------	--

Definition at line 1279 of file specfun.h.

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

Return the logarithmic integral of argument x .

See also

[logint](#) for details.

Definition at line 1255 of file specfun.h.

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

Return the logarithmic integral of argument x .

See also

[logint](#) for details.

Definition at line 1264 of file specfun.h.

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

Definition at line 2767 of file specfun.h.

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

Definition at line 2755 of file specfun.h.

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

Definition at line 2759 of file specfun.h.

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

Definition at line 2746 of file specfun.h.

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

Definition at line 2734 of file specfun.h.

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

Definition at line 2738 of file specfun.h.

6.1.3.202 `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 of thing1 `h` and thing2 `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 4088 of file specfun.h.

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

Return the Owens T function function of thing `h` and argument `a`.

See also

[owens_t](#) for details.

Definition at line 4062 of file specfun.h.

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

Return the Owens T function function of thing `h` and argument `a`.

See also

[owens_t](#) for details.

Definition at line 4071 of file specfun.h.

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

Definition at line 2809 of file specfun.h.

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

Definition at line 2797 of file specfun.h.

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

Definition at line 2801 of file specfun.h.

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

Definition at line 2788 of file specfun.h.

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

Definition at line 2776 of file specfun.h.

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

Definition at line 2780 of file specfun.h.

6.1.3.211 `template<typename _Tp > std::complex<__gnu_cxx::__promote_num_t<_Tp> > __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 3500 of file specfun.h.

6.1.3.212 `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 3473 of file specfun.h.

6.1.3.213 `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 3483 of file `specfun.h`.

6.1.3.214 `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 2511 of file `specfun.h`.

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

Definition at line 2492 of file `specfun.h`.

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

Definition at line 2496 of file `specfun.h`.

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

Definition at line 1824 of file `specfun.h`.

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

Definition at line 1812 of file `specfun.h`.

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

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

Definition at line 1816 of file `specfun.h`.

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

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

Definition at line 1241 of file `specfun.h`.

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

Definition at line 1223 of file specfun.h.

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

Definition at line 1211 of file specfun.h.

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

Definition at line 1215 of file specfun.h.

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

Definition at line 1232 of file specfun.h.

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

Definition at line 1236 of file specfun.h.

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

Definition at line 1866 of file specfun.h.

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

Definition at line 1845 of file specfun.h.

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

Definition at line 1833 of file specfun.h.

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

Definition at line 1837 of file specfun.h.

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

Definition at line 1854 of file specfun.h.

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

Definition at line 1858 of file specfun.h.

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

Return the hyperbolic sine integral of argument x .

The sine hyperbolic integral is defined by

Parameters

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

Definition at line 1393 of file `specfun.h`.

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

Return the hyperbolic sine integral of argument x .

See also

[sinhint](#) for details.

Definition at line 1369 of file `specfun.h`.

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

Return the hyperbolic sine integral of argument x .

See also

[sinhint](#) for details.

Definition at line 1378 of file `specfun.h`.

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

Return the sine integral of argument x .

The sine integral is defined by

Parameters

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

Definition at line 1317 of file `specfun.h`.

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

Return the sine integral of argument x .

See also

[sinint](#) for details.

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

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

Return the sine integral of argument x .

See also

[sinint](#) for details.

Definition at line 1302 of file `specfun.h`.

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

Definition at line 1981 of file `specfun.h`.

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

Definition at line 1959 of file `specfun.h`.

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

Definition at line 1968 of file `specfun.h`.

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

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

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

Definition at line 1993 of file `specfun.h`.

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

Definition at line 2002 of file `specfun.h`.

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

Definition at line 1929 of file `specfun.h`.

6.1.3.245 `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 of non-negative order n and complex argument 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 integral order ≥ 0
<code>__x</code>	The complex argument

Definition at line 3378 of file `specfun.h`.

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

Definition at line 1917 of file `specfun.h`.

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

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

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

Definition at line 1921 of file `specfun.h`.

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

Definition at line 3361 of file `specfun.h`.

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

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

6.1.3.251 `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 of non-negative order n and complex argument 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 integral order ≥ 0
<code>__x</code>	The complex argument

Definition at line 3408 of file `specfun.h`.

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

Definition at line 1938 of file `specfun.h`.

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

Definition at line 3387 of file specfun.h.

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

Definition at line 1942 of file specfun.h.

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

Definition at line 3391 of file specfun.h.

6.1.3.256 `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 real 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 3458 of file specfun.h.

6.1.3.257 `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 real zenith angle θ , and real azimuth angle ϕ .

See also

[sph_harmonic](#) for details.

Definition at line 3423 of file specfun.h.

6.1.3.258 `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 real zenith angle θ , and real azimuth angle ϕ .

See also

[sph_harmonic](#) for details.

Definition at line 3434 of file specfun.h.

```
6.1.3.259 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 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 3744 of file specfun.h.

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

Return the exponential theta-1 function of period `nu` and argument `x`.

See also

[theta_1](#) for details.

Definition at line 3718 of file specfun.h.

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

Return the exponential theta-1 function of period `nu` and argument `x`.

See also

[theta_1](#) for details.

Definition at line 3727 of file specfun.h.

```
6.1.3.262 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 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 3784 of file `specfun.h`.

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

Return the exponential theta-2 function of period `nu` and argument `x`.

See also

[theta_2](#) for details.

Definition at line 3758 of file `specfun.h`.

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

Return the exponential theta-2 function of period `nu` and argument `x`.

See also

[theta_2](#) for details.

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

6.1.3.265 `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 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 3824 of file `specfun.h`.

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

Return the exponential theta-3 function of period `nu` and argument `x`.

See also

[theta_3](#) for details.

Definition at line 3798 of file `specfun.h`.

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

Return the exponential theta-3 function of period `nu` and argument `x`.

See also

[theta_3](#) for details.

Definition at line 3807 of file `specfun.h`.

6.1.3.268 `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 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 3864 of file `specfun.h`.

6.1.3.269 `float __gnu_cxx::theta_4f (float __nu, float __x) [inline]`

Return the exponential theta-4 function of period `nu` and argument `x`.

See also

[theta_4](#) for details.

Definition at line 3838 of file `specfun.h`.

6.1.3.270 `long double __gnu_cxx::theta_4l (long double __nu, long double __x) [inline]`

Return the exponential theta-4 function of period `nu` and argument `x`.

See also

[theta_4](#) for details.

Definition at line 3847 of file `specfun.h`.

6.1.3.271 `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 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 3970 of file specfun.h.

6.1.3.272 `float __gnu_cxx::theta_cf(float __k, float __x) [inline]`

Return the Neville theta-c function of modulus k and argument x .

See also

[theta_c](#) for details.

Definition at line 3945 of file specfun.h.

6.1.3.273 `long double __gnu_cxx::theta_cl(long double __k, long double __x) [inline]`

Return the Neville theta-c function of modulus k and argument x .

See also

[theta_c](#) for details.

Definition at line 3954 of file specfun.h.

6.1.3.274 `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 of modulus k and argument x .

The Neville theta-d function is defined by

Parameters

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

Definition at line 4009 of file specfun.h.

6.1.3.275 `float __gnu_cxx::theta_df(float __k, float __x) [inline]`

Return the Neville theta-d function of modulus k and argument x .

See also

[theta_d](#) for details.

Definition at line 3984 of file specfun.h.

6.1.3.276 `long double __gnu_cxx::theta_dl (long double __k, long double __x) [inline]`

Return the Neville theta-d function of modulus k and argument x .

See also

[theta_d](#) for details.

Definition at line 3993 of file specfun.h.

6.1.3.277 `template<typename _Tp, typename _Tp> __gnu_cxx::__promote_num_t<_Tp, _Tp> __gnu_cxx::theta_n (_Tp __k, _Tp __x) [inline]`

Return the Neville theta-n function of modulus k and argument x .

The Neville theta-n function is defined by

Parameters

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

Definition at line 4048 of file specfun.h.

6.1.3.278 `float __gnu_cxx::theta_nf (float __k, float __x) [inline]`

Return the Neville theta-n function of modulus k and argument x .

See also

[theta_n](#) for details.

Definition at line 4023 of file specfun.h.

6.1.3.279 `long double __gnu_cxx::theta_nl (long double __k, long double __x) [inline]`

Return the Neville theta-n function of modulus k and argument x .

See also

[theta_n](#) for details.

Definition at line 4032 of file specfun.h.

6.1.3.280 `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 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 3931 of file specfun.h.

6.1.3.281 `float __gnu_cxx::theta_sf(float __k, float __x) [inline]`

Return the Neville theta-s function of modulus k and argument x .

See also

[theta_s](#) for details.

Definition at line 3906 of file specfun.h.

6.1.3.282 `long double __gnu_cxx::theta_sl(long double __k, long double __x) [inline]`

Return the Neville theta-s function of modulus k and argument x .

See also

[theta_s](#) for details.

Definition at line 3915 of file specfun.h.

6.1.3.283 `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]`

Definition at line 1803 of file specfun.h.

6.1.3.284 `float __gnu_cxx::zernikef(unsigned int __n, int __m, float __rho, float __phi) [inline]`

Definition at line 1791 of file specfun.h.

6.1.3.285 `long double __gnu_cxx::zernikel(unsigned int __n, int __m, long double __rho, long double __phi) [inline]`

Definition at line 1795 of file specfun.h.

6.2 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)`

6.2.1 Detailed Description

A collection of advanced mathematical special functions.

6.2.2 Function Documentation

6.2.2.1 `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 of order n , degree m : $L_n^m(x)$.

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:

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

Parameters

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

Definition at line 161 of file specfun.h.

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

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

See also

[assoc_laguerre](#) for more details.

Definition at line 119 of file specfun.h.

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

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

See also

[assoc_laguerre](#) for more details.

Definition at line 129 of file specfun.h.

6.2.2.4 `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 of degree l and order 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 degree 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 206 of file specfun.h.

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

Return the associated Legendre function of degree `l` and order `m` for `float` argument.

See also

[assoc_legendre](#) for more details.

Definition at line 176 of file specfun.h.

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

Return the associated Legendre function of degree `l` and order `m`.

See also

[assoc_legendre](#) for more details.

Definition at line 185 of file specfun.h.

6.2.2.7 `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

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

Parameters

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

Definition at line 247 of file specfun.h.

6.2.2.8 `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 220 of file specfun.h.

6.2.2.9 `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 230 of file specfun.h.

6.2.2.10 `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.

Parameters

<code>__k</code>	The modulus
------------------	-------------

Definition at line 291 of file specfun.h.

6.2.2.11 `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 262 of file specfun.h.

6.2.2.12 `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 272 of file specfun.h.

6.2.2.13 `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

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

Parameters

<code>__k</code>	The modulus
------------------	-------------

Definition at line 332 of file `specfun.h`.

6.2.2.14 `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 306 of file `specfun.h`.

6.2.2.15 `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 316 of file `specfun.h`.

6.2.2.16 `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

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

Parameters

<code>__k</code>	The modulus of the elliptic function.
------------------	---------------------------------------

<code>__nu</code>	The argument of the elliptic function.
-------------------	--

Definition at line 376 of file specfun.h.

6.2.2.17 `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 347 of file specfun.h.

6.2.2.18 `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 357 of file specfun.h.

6.2.2.19 `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 $I_\nu(x)$ of real order ν and argument 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.

Definition at line 419 of file specfun.h.

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

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

See also

[cyl_bessel_i](#) for details.

Definition at line 391 of file specfun.h.

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

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

See also

[cyl_bessel_i](#) for setails.

Definition at line 401 of file `specfun.h`.

6.2.2.22 `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 .

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.

Definition at line 462 of file `specfun.h`.

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

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

See also

[cyl_bessel_j](#) for setails.

Definition at line 434 of file `specfun.h`.

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

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

See also

[cyl_bessel_j](#) for setails.

Definition at line 444 of file `specfun.h`.

6.2.2.25 `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:

$$K_{-\nu}(x) = K_\nu(x)$$

Parameters

<code>__nu</code>	The order of the irregular modified Bessel function.
<code>__x</code>	The argument of the irregular modified Bessel function.

Definition at line 511 of file specfun.h.

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

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

See also

[cyl_bessel_k](#) for details.

Definition at line 477 of file specfun.h.

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

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

See also

[cyl_bessel_k](#) for details.

Definition at line 487 of file specfun.h.

6.2.2.28 `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 .

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.

Definition at line 556 of file specfun.h.

6.2.2.29 `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 526 of file `specfun.h`.

6.2.2.30 `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 536 of file `specfun.h`.

6.2.2.31 `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](#).

Parameters

<code>__k</code>	The modulus of the elliptic function.
<code>__phi</code>	The integral limit argument of the elliptic function.

Definition at line 601 of file `specfun.h`.

6.2.2.32 `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 571 of file `specfun.h`.

6.2.2.33 `long double std::ellint_1(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 581 of file `specfun.h`.

6.2.2.34 `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](#).

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 646 of file `specfun.h`.

6.2.2.35 `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 616 of file `specfun.h`.

6.2.2.36 `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 626 of file `specfun.h`.

6.2.2.37 `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](#).

Parameters

<code>__k</code>	The modulus 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 695 of file `specfun.h`.

6.2.2.38 `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 661 of file `specfun.h`.

6.2.2.39 `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 671 of file `specfun.h`.

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

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

Definition at line 734 of file specfun.h.

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

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

See also

[expint](#) for details.

Definition at line 709 of file specfun.h.

6.2.2.42 `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 719 of file specfun.h.

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

Return the Hermite polynomial of order n , $H_n(x)$, for `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:

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

Parameters

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

Definition at line 781 of file specfun.h.

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

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

See also

[hermite](#) for details.

Definition at line 749 of file specfun.h.

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

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

See also

[hermite](#) for details.

Definition at line 759 of file specfun.h.

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

Returns the Laguerre polynomial of degree n , and argument x : $L_n(x)$.

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 function.
<code>__x</code>	The argument of the Laguerre function.

Definition at line 823 of file specfun.h.

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

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

See also

[laguerre](#) for more details.

Definition at line 796 of file specfun.h.

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

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

See also

[laguerre](#) for more details.

Definition at line 806 of file specfun.h.

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

Return the Legendre polynomial $P_l(x)$ of degree l for `real` argument.

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 degree $l \geq 0$
<code>__x</code>	The argument $ x \leq 1$

Definition at line 866 of file specfun.h.

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

Return the Legendre polynomial $P_l(x)$ of degree l for `float` argument.

See also

[legendre](#) for more details.

Definition at line 838 of file specfun.h.

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

Return the Legendre polynomial $P_l(x)$ of degree l for `long double` argument.

See also

[legendre](#) for more details.

Definition at line 848 of file specfun.h.

6.2.2.52 `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:

$$\zeta(s) = \sum_{k=1}^{\infty} k^{-s} \text{ for } s > 1 \quad \frac{(2\pi)^s}{\pi i} \sin\left(\frac{\pi s}{2}\right) \Gamma(1-s) \zeta(1-s) \text{ for } s < 1$$

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 $s \neq 1$
------------------	-------------------------

Definition at line 913 of file specfun.h.

6.2.2.53 `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 881 of file specfun.h.

6.2.2.54 `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 891 of file specfun.h.

6.2.2.55 `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 order n for real argument $x \geq 0$.

The spherical Bessel function is defined by:

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

Parameters

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

Definition at line 955 of file specfun.h.

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

Return the spherical Bessel function $j_n(x)$ of order n for `float` argument.

See also

[sph_bessel](#) for more details.

Definition at line 928 of file specfun.h.

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

Return the spherical Bessel function $j_n(x)$ of order n for `long double` argument.

See also

[sph_bessel](#) for more details.

Definition at line 938 of file specfun.h.

6.2.2.58 `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 non-negative integral degree l and order m and real angle θ in radians.

The spherical Legendre function is defined by

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

Parameters

<code>__l</code>	The non-negative order $l \geq 0$.
<code>__m</code>	The non-negative degree $m \geq 0$ and $m \leq l$.
<code>__theta</code>	The radian polar angle argument

Definition at line 1001 of file specfun.h.

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

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

See also

[sph_legendre](#) for details.

Definition at line 970 of file specfun.h.

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

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

See also

[sph_legendre](#) for details.

Definition at line 981 of file specfun.h.

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

Return the spherical Neumann function of non-negative integral 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 non-negative integral order
<code>__x</code>	The non-negative real argument

Definition at line 1043 of file specfun.h.

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

Return the spherical Neumann function of non-negative integral order `n` and non-negative float argument `x`.

See also

[sph_neumann](#) for details.

Definition at line 1016 of file specfun.h.

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

Return the spherical Neumann function of non-negative integral order `n` and non-negative `long double` argument `x`.

See also

[sph_neumann](#) for details.

Definition at line 1026 of file `specfun.h`.

Chapter 7

Namespace Documentation

7.1 `__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 __x)`
- `float dirichlet_betaf (float __x)`
- `long double dirichlet_betall (long double __x)`
- `template<typename _Tp >`
`_Tp dirichlet_eta (_Tp __x)`
- `float dirichlet_etaf (float __x)`
- `long double dirichlet_etall (long double __x)`
- `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_dfl (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)
- float [hurwitz_zetaf](#) (float __s, float __a)
- long double [hurwitz_zetal](#) (long double __s, long double __a)
- template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >
__gnu_cxx::__promote_4< _Tpa, _Tpb, _Tpc, _Tp >::__type [hyperg](#) (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)
- float [hypergf](#) (float __a, float __b, float __c, float __x)
- long double [hypergl](#) (long double __a, long double __b, long double __c, long double __x)
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > [ibeta](#) (_Ta __a, _Tb __b, _Tp __x)
- template<typename _Ta, typename _Tb, typename _Tp >
__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > [ibetac](#) (_Ta __a, _Tb __b, _Tp __x)
- float [ibetacf](#) (float __a, float __b, float __x)
- long double [ibetacdl](#) (long double __a, long double __b, long double __x)
- float [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 [lbincoefl](#) (unsigned int __n, unsigned int __k)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [ldouble_factorial](#) (int __n)
- float [ldouble_factorialf](#) (int __n)
- long double [ldouble_factoriall](#) (int __n)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [legendre_q](#) (unsigned int __n, _Tp __x)
- float [legendre_qf](#) (unsigned int __n, float __x)
- long double [legendre_ql](#) (unsigned int __n, long double __x)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [lfactorial](#) (unsigned int __n)
- float [lfactorialf](#) (unsigned int __n)
- long double [lfactoriall](#) (unsigned int __n)
- template<typename _Tp >
__gnu_cxx::__promote_num_t< _Tp > [logint](#) (_Tp __x)
- float [logintf](#) (float __x)
- long double [logintl](#) (long double __x)
- template<typename _Tp, typename _Tn >
__gnu_cxx::__promote_num_t< _Tp, _Tn > [lpochhammer_l](#) (_Tp __a, _Tn __n)
- float [lpochhammer_lf](#) (float __a, float __n)
- long double [lpochhammer_ll](#) (long double __a, long double __n)
- template<typename _Tp, typename _Tn >
__gnu_cxx::__promote_num_t< _Tp, _Tn > [lpochhammer_u](#) (_Tp __a, _Tn __n)
- float [lpochhammer_uf](#) (float __a, float __n)
- long double [lpochhammer_ul](#) (long double __a, long double __n)
- template<typename _Tph, typename _Tpa >
__gnu_cxx::__promote_num_t< _Tph, _Tpa > [owens_t](#) (_Tph __h, _Tpa __a)
- float [owens_tf](#) (float __h, float __a)
- long double [owens_tl](#) (long double __h, long double __a)
- template<typename _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 >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > polylog (_Tp __s, std::complex< _Tp > __w)`
- `std::complex< float > polylogf (float __s, std::complex< float > __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`)

7.2 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)`
- `float ellint_3f (float __k, float __nu, float __phi)`
Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$.
- `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)`

7.3 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 $\text{abs}(z)$.*
- 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 > &_Ip1d3, std::complex< _Tp > &_Im1d3, std::complex< _Tp > &_Ip2d3, std::complex< _Tp > &_Im2d3)
Compute approximations to the modified Bessel functions of the second kind of orders 1/3 and 2/3 for moderate arguments.
- 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

$$\frac{\Gamma(\nu + 1)}{(z/2)^{\nu}} I_{\nu}(z) = {}_0F_1(; \nu + 1; z^2/4),$$

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.

- `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_steud (_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_const_frac(_Tp __x)`
Compute the Dawson integral using a sampling theorem representation.
- `template<typename _Tp >`
`_Tp __dawson_series(_Tp __x)`
Compute the Dawson integral using the series expansion.
- `template<typename _Tp >`
`void __debye_region(std::complex<_Tp> __alpha, int &__indexr, char &__aorb)`
- `template<typename _Tp >`
`_Tp __dilog(_Tp __x)`
Compute the dilogarithm function $Li_2(x)$ by summation for $x \leq 1$.

- `template<typename _Tp >`
`_Tp __dirichlet_beta (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp __dirichlet_beta (_Tp __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __dirichlet_eta (std::complex< _Tp > __w)`
- `template<typename _Tp >`
`_Tp __dirichlet_eta (_Tp __w)`
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __double_factorial (int __n)`
Return the double factorial of the integer n .
- `template<typename _Tp >`
`_Tp __ellint_1 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the first kind $F(k, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_2 (_Tp __k, _Tp __phi)`
Return the incomplete elliptic integral of the second kind $E(k, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_3 (_Tp __k, _Tp __nu, _Tp __phi)`
Return the incomplete elliptic integral of the third kind $\Pi(k, \nu, \phi)$ using the Carlson formulation.
- `template<typename _Tp >`
`_Tp __ellint_cel (_Tp __k_c, _Tp __p, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp __ellint_d (_Tp __k, _Tp __phi)`
- `template<typename _Tp >`
`_Tp __ellint_el1 (_Tp __x, _Tp __k_c)`
- `template<typename _Tp >`
`_Tp __ellint_el2 (_Tp __x, _Tp __k_c, _Tp __a, _Tp __b)`
- `template<typename _Tp >`
`_Tp __ellint_el3 (_Tp __x, _Tp __k_c, _Tp __p)`
- `template<typename _Tp >`
`_Tp __ellint_rc (_Tp __x, _Tp __y)`
Return the Carlson elliptic function $R_C(x, y) = R_F(x, y, y)$ where $R_F(x, y, z)$ is the Carlson elliptic function of the first kind.
- `template<typename _Tp >`
`_Tp __ellint_rd (_Tp __x, _Tp __y, _Tp __z)`
Return the Carlson elliptic function of the second kind $R_D(x, y, z) = R_J(x, y, z, z)$ where $R_J(x, y, z, p)$ is the Carlson elliptic function of the third kind.
- `template<typename _Tp >`
`_Tp __ellint_rf (_Tp __x, _Tp __y, _Tp __z)`
Return the Carlson elliptic function $R_F(x, y, z)$ of the first kind.
- `template<typename _Tp >`
`_Tp __ellint_rg (_Tp __x, _Tp __y, _Tp __z)`
Return the symmetric Carlson elliptic function of the second kind $R_G(x, y, z)$.
- `template<typename _Tp >`
`_Tp __ellint_rj (_Tp __x, _Tp __y, _Tp __z, _Tp __p)`
Return the Carlson elliptic function $R_J(x, y, z, p)$ of the third kind.
- `template<typename _Tp >`
`_Tp __ellnome (_Tp __k)`
- `template<typename _Tp >`
`_Tp __ellnome_k (_Tp __k)`

- `template<typename _Tp >`
`_Tp __ellnome_series (_Tp __k)`
- `template<typename _Tp >`
`_Tp __expint (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$.
- `template<typename _Tp >`
`_Tp __expint (_Tp __x)`
Return the exponential integral $Ei(x)$.
- `template<typename _Tp >`
`_Tp __expint_asymp (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ for large argument.
- `template<typename _Tp >`
`_Tp __expint_E1 (_Tp __x)`
Return the exponential integral $E_1(x)$.
- `template<typename _Tp >`
`_Tp __expint_E1_asymp (_Tp __x)`
Return the exponential integral $E_1(x)$ by asymptotic expansion.
- `template<typename _Tp >`
`_Tp __expint_E1_series (_Tp __x)`
Return the exponential integral $E_1(x)$ by series summation. This should be good for $x < 1$.
- `template<typename _Tp >`
`_Tp __expint_Ei (_Tp __x)`
Return the exponential integral $Ei(x)$.
- `template<typename _Tp >`
`_Tp __expint_Ei_asymp (_Tp __x)`
Return the exponential integral $Ei(x)$ by asymptotic expansion.
- `template<typename _Tp >`
`_Tp __expint_Ei_series (_Tp __x)`
Return the exponential integral $Ei(x)$ by series summation.
- `template<typename _Tp >`
`_Tp __expint_En_cont_frac (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ by continued fractions.
- `template<typename _Tp >`
`_Tp __expint_En_recursion (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ by recursion. Use upward recursion for $x < n$ and downward recursion (Miller's algorithm) otherwise.
- `template<typename _Tp >`
`_Tp __expint_En_series (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ by series summation.
- `template<typename _Tp >`
`_Tp __expint_large_n (unsigned int __n, _Tp __x)`
Return the exponential integral $E_n(x)$ for large order.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp __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 __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 __factorial (unsigned int __n)`
Return the factorial of the integer n .
- `template<typename _Tp >`
`_Tp __fermi_dirac (_Tp __s, _Tp __x)`
- `template<typename _Tp >`
`void __fock_airy (_Tp __x, std::complex< _Tp > &__w1, std::complex< _Tp > &__w2, std::complex< _Tp > &__w1p, std::complex< _Tp > &__w2p)`
Compute the Fock-type Airy functions $w_1(x)$ and $w_2(x)$ and their first derivatives $w_1'(x)$ and $w_2'(x)$ respectively.

$$w_1(x) = \sqrt{\pi}(Ai(x) + iBi(x))$$

$$w_2(x) = \sqrt{\pi}(Ai(x) - iBi(x))$$
- `template<typename _Tp >`
`bool __fpequal (const _Tp &__a, const _Tp &__b)`
- `template<typename _Tp >`
`bool __fpimag (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`
`bool __fpimag (const _Tp)`
- `template<typename _Tp >`
`bool __fpreal (const std::complex< _Tp > &__w)`
- `template<typename _Tp >`
`bool __fpreal (const _Tp)`
- `template<typename _Tp >`
`std::complex< _Tp > __fresnel (const _Tp __x)`
Return the Fresnel cosine and sine integrals as a complex number $\$f[C(x) + iS(x)]$.
- `template<typename _Tp >`
`void __fresnel_cont_frac (const _Tp __ax, _Tp &__Cf, _Tp &__Sf)`
This function computes the Fresnel cosine and sine integrals by continued fractions for positive argument.
- `template<typename _Tp >`
`void __fresnel_series (const _Tp __ax, _Tp &__Cf, _Tp &__Sf)`
This function returns the Fresnel cosine and sine integrals as a pair by series expansion for positive argument.
- `template<typename _Tp >`
`_Tp __gamma (_Tp __x)`
Return $\Gamma(x)$.
- `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> &__zetat)

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 >`
`_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 [__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, $(n)_n = n!$. 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 __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 __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

$$(a)_n = \prod_{k=0}^{n-1} (a + k), (a)_0 = 1 = \Gamma(a + n) / \Gamma(n)$$

Many notations exist:

$$a^{\overline{n}}$$

,

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

, 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_asyp (unsigned int __n, _Tp __x)`
This routine returns the Hermite polynomial of large order n : $H_n(x)$. We assume here that $x \geq 0$.
- `template<typename _Tp >`
`_Tp __poly_hermite_recursion (unsigned int __n, _Tp __x)`
This routine returns the Hermite polynomial of order n : $H_n(x)$ by recursion on n .
- `template<typename _Tp >`
`_Tp __poly_jacobi (unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)`
- `template<typename _Tpa, typename _Tp >`
`_Tp __poly_laguerre (unsigned int __n, _Tpa __alpha1, _Tp __x)`
This routine returns the associated Laguerre polynomial of order n , degree α : $L_n^\alpha(x)$.
- `template<typename _Tpa, typename _Tp >`
`_Tp __poly_laguerre_hyperg (unsigned int __n, _Tpa __alpha1, _Tp __x)`
Evaluate the polynomial based on the confluent hypergeometric function in a safe way, with no restriction on the arguments.
- `template<typename _Tpa, typename _Tp >`
`_Tp __poly_laguerre_large_n (unsigned __n, _Tpa __alpha1, _Tp __x)`
This routine returns the associated Laguerre polynomial of order n , degree α 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_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 __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_asyp (const _Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_int_neg (const 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 (const unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > __polylog_exp_int_pos (const 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

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

- `template<typename _Tp >`
`std::pair<_Tp, _Tp> __sincosint (_Tp __x)`

This function returns the sine $Si(x)$ and cosine $Chi(x)$ 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 $Chi(x)$ 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 $Chi(x)$ 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 $Chi(x)$ 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

$$\sinhc_a(x) = \frac{\sinh(\pi x/a)}{\pi x/a}$$

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __sinhc (_Tp __x)`

Return the normalized hyperbolic sinus cardinal function

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

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __sinhc_pi (_Tp __x)`

Return the unnormalized hyperbolic sinus cardinal function

$$\sinhc_\pi(x) = \frac{\sinh(x)}{x}$$

- `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 [__students_t_cdf](#) (_Tp __t, unsigned int __nu)

Return the Students T probability function.

- template<typename _Tp >
_GLIBCXX14_CONSTEXPR _Tp [__students_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](#)]

7.3.1 Enumeration Type Documentation

7.3.1.1 anonymous enum

Enumerator

SININT

COSINT

Definition at line 42 of file `sf_trigint.tcc`.

7.3.2 Function Documentation

7.3.2.1 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 486 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`, and `__cyl_bessel_jn()`.

7.3.2.2 `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

and

See also

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 1008 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()`.

7.3.2.3 `template<typename _Tp> std::complex<_Tp> std::__detail::__airy_ai (std::complex<_Tp> __z)`

Return the complex Airy Ai function.

Definition at line 1145 of file sf_airy.tcc.

References `__airy()`.

7.3.2.4 `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()`.

7.3.2.5 `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 the sake of speed and the fact that this function is to be 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()`.

7.3.2.6 `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 the sake of speed and the fact that this function is to be called by another, checks for valid arguments are not made. Hence, an error return is not needed. 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 187 of file `sf_airy.tcc`.

Referenced by `__airy()`.

7.3.2.7 `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 the sake of 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 $I_{(-1/3)}(z)$.
out	<code>_lm1d3</code>	The value of $I_{(-1/3)}(z)$.
out	<code>_lp2d3</code>	The value of $I_{(2/3)}(z)$.
out	<code>_lm2d3</code>	The value of $I_{(2/3)}(z)$.

Definition at line 393 of file `sf_airy.tcc`.

Referenced by `__airy()`.

7.3.2.8 `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 607 of file `sf_airy.tcc`.

Referenced by `__airy()`.

7.3.2.9 `template<typename _Tp> std::complex<_Tp> std::__detail::__airy_bi (std::complex<_Tp> __z)`

Return the complex Airy Bi function.

Definition at line 1158 of file `sf_airy.tcc`.

References `__airy()`.

7.3.2.10 `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 the sake of speed and the fact that this function will be driven 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 791 of file `sf_airy.tcc`.

Referenced by `__airy()`.

7.3.2.11 `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:

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

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 292 of file `sf_laguerre.tcc`.

Referenced by `__hydrogen()`.

7.3.2.12 `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()`.

7.3.2.13 `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.

7.3.2.14 `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.

7.3.2.15 `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.

7.3.2.16 `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()`.

7.3.2.17 `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()`.

7.3.2.18 `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()`, `__f_cdf()`, `__f_cdfc()`, `__students_t_cdf()`, and `__students_t_cdfc()`.

7.3.2.19 `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()`.

7.3.2.20 `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()`.

7.3.2.21 `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.

7.3.2.22 `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.

7.3.2.23 `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 404 of file sf_beta.tcc.

References `__beta_inc()`.

7.3.2.24 `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 434 of file `sf_beta.tcc`.

References `__beta_inc()`.

7.3.2.25 `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 <code>s</code> ≥ 0 .
<code>__x</code>	The real argument.

Returns

The real Fermi-Dirac cosine sum `G_s(x)`,

Definition at line 1369 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

7.3.2.26 `template<typename _Tp> _Tp std::__detail::__chebyshev_recur (unsigned int __n, _Tp __x, _Tp __C0, _Tp __C1)`

Definition at line 44 of file `sf_chebyshev.tcc`.

Referenced by `__chebyshev_t()`, `__chebyshev_u()`, `__chebyshev_v()`, and `__chebyshev_w()`.

7.3.2.27 `template<typename _Tp> _Tp std::__detail::__chebyshev_t (unsigned int __n, _Tp __x)`

Definition at line 58 of file `sf_chebyshev.tcc`.

References `__chebyshev_recur()`.

7.3.2.28 `template<typename _Tp> _Tp std::__detail::__chebyshev_u (unsigned int __n, _Tp __x)`

Definition at line 73 of file `sf_chebyshev.tcc`.

References `__chebyshev_recur()`.

7.3.2.29 `template<typename _Tp> _Tp std::__detail::__chebyshev_v (unsigned int __n, _Tp __x)`

Definition at line 88 of file `sf_chebyshev.tcc`.

References `__chebyshev_recur()`.

7.3.2.30 `template<typename _Tp> _Tp std::__detail::__chebyshev_w (unsigned int __n, _Tp __x)`

Definition at line 103 of file `sf_chebyshev.tcc`.

References `__chebyshev_recur()`.

7.3.2.31 `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()`.

7.3.2.32 `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()`.

7.3.2.33 `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()`.

7.3.2.34 `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()`.

7.3.2.35 `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()`.

7.3.2.36 `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()`.

7.3.2.37 `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()`.

7.3.2.38 `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 1198 of file sf_polylog.tcc.

References `__polylog_exp()`.

7.3.2.39 `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 1222 of file sf_polylog.tcc.

References `__polylog_exp()`.

7.3.2.40 `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 1297 of file sf_polylog.tcc.

References `__polylog_exp()`.

7.3.2.41 `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 1322 of file sf_polylog.tcc.

References `__polylog_exp()`.

7.3.2.42 `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 1247 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

7.3.2.43 `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 1272 of file `sf_polylog.tcc`.

References `__polylog_exp()`.

7.3.2.44 `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()`.

7.3.2.45 `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

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

Parameters

<code>__k</code>	The modulus of the complete elliptic function.
------------------	--

Returns

The complete elliptic function of the second kind.

Definition at line 638 of file sf_ellint.tcc.

References `__ellint_rd()`, and `__ellint_rf()`.

Referenced by `__ellint_2()`.

7.3.2.46 `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()`.

7.3.2.47 `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()`.

7.3.2.48 `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()`.

7.3.2.49 `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()`.

7.3.2.50 `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()`.

7.3.2.51 `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()`.

7.3.2.52 `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()`.

7.3.2.53 `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()`.

7.3.2.54 `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()`.

7.3.2.55 `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()`.

7.3.2.56 `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

<code>in</code>	<code>__nu</code>	The order for which the cylindrical Bessel function is evaluated.
<code>in</code>	<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()`.

7.3.2.57 `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 375 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_ik()`.

7.3.2.58 `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()`.

7.3.2.59 `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()`.

7.3.2.60 `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_steel()`.

7.3.2.61 `template<typename _Tp > void std::__detail::__cyl_bessel_ik_steel (_Tp __nu, _Tp __x, _Tp & _Inu, _Tp & _Knu, _Tp & _Ipnu, _Tp & _Kpnu)`

Compute the modified Bessel functions $I_\nu(x)$ and $K_\nu(x)$ and their first derivatives $I'_\nu(x)$ and $K'_\nu(x)$ respectively. These four functions are computed together for numerical stability.

Parameters

<code>__nu</code>	The order of the Bessel functions.
<code>__x</code>	The argument of the Bessel functions.
<code>_Inu</code>	The output regular modified Bessel function.
<code>_Knu</code>	The output irregular modified Bessel function.
<code>_Ipnu</code>	The output derivative of the regular modified Bessel function.
<code>_Kpnu</code>	The output derivative of the irregular modified Bessel function.

Definition at line 152 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik_asymp()`, and `__gamma_temme()`.

Referenced by `__cyl_bessel_ik()`.

7.3.2.62 `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 503 of file `sf_bessel.tcc`.

References `__cyl_bessel_ij_series()`, and `__cyl_bessel_jn()`.

7.3.2.63 `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_steeds()`.

Referenced by `__airy()`, `__cyl_bessel_j()`, `__cyl_hankel_1()`, `__cyl_hankel_2()`, `__cyl_neumann_n()`, and `__sph_bessel_jn()`.

7.3.2.64 `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_steeds()`.

7.3.2.65 `template<typename _Tp> void std::__detail::__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.

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()`.

7.3.2.66 `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 413 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`.

7.3.2.67 `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 568 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

7.3.2.68 `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()`.

7.3.2.69 `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 604 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

7.3.2.70 `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_hankel_2 (std::complex<_Tp> __nu, std::complex<_Tp> __z)`

Return the complex cylindrical Hankel function of the second kind.

Parameters

in	<code>__nu</code>	The order for which the cylindrical Hankel function of the second kind is evaluated.
in	<code>__z</code>	The argument at which the cylindrical Hankel function of the second kind is evaluated.

Returns

The complex cylindrical Hankel function of the second kind.

Definition at line 1206 of file sf_hankel.tcc.

References `__hankel()`.

7.3.2.71 `template<typename _Tp> std::complex<_Tp> std::__detail::__cyl_neumann (std::complex<_Tp> __nu, std::complex<_Tp> __z)`

Return the complex cylindrical Neumann function.

Parameters

in	<code>__nu</code>	The order for which the cylindrical Neumann function is evaluated.
in	<code>__z</code>	The argument at which the cylindrical Neumann function is evaluated.

Returns

The complex cylindrical Neumann function.

Definition at line 1238 of file sf_hankel.tcc.

References `__hankel()`.

7.3.2.72 `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 538 of file sf_bessel.tcc.

References `__cyl_bessel_jn()`.

7.3.2.73 `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 $-inf < x < inf$.
------------------	---------------------------------

Definition at line 233 of file `sf_dawson.tcc`.

References `__dawson_const_frac()`, and `__dawson_series()`.

7.3.2.74 `template<typename _Tp> _Tp std::__detail::__dawson_const_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()`.

7.3.2.75 `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()`.

7.3.2.76 `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()`.

7.3.2.77 `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.

7.3.2.78 `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 1160 of file sf_polylog.tcc.

References `__fpequal()`, and `__polylog()`.

7.3.2.79 `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 1179 of file sf_polylog.tcc.

References `__polylog()`.

7.3.2.80 `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 1123 of file sf_polylog.tcc.

References `__fpequal()`, and `__polylog()`.

7.3.2.81 `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 1141 of file sf_polylog.tcc.

References `__polylog()`.

7.3.2.82 `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`.

7.3.2.83 `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()`.

7.3.2.84 `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()`.

7.3.2.85 `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()`.

7.3.2.86 `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()`.

7.3.2.87 `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()`.

7.3.2.88 `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()`.

7.3.2.89 `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()`.

7.3.2.90 `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()`.

7.3.2.91 `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()`.

7.3.2.92 `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()`.

7.3.2.93 `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()`.

7.3.2.94 `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()`.

7.3.2.95 `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()`.

7.3.2.96 `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()`.

7.3.2.97 `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()`.

7.3.2.98 `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()`.

7.3.2.99 `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()`.

7.3.2.100 `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()`.

7.3.2.101 `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.

7.3.2.102 `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()`.

7.3.2.103 `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()`.

7.3.2.104 `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()`.

7.3.2.105 `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_asymp()`, and `__expint_Ei_series()`.

Referenced by `__coshint()`, `__expint()`, `__expint_E1()`, and `__sinhint()`.

7.3.2.106 `template<typename _Tp> _Tp std::__detail::__expint_Ei_asymp (_Tp __x)`

Return the exponential integral $Ei(x)$ by asymptotic expansion.

The exponential integral is given by

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

Parameters

<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()`.

7.3.2.107 `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()`.

7.3.2.108 `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()`.

7.3.2.109 `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()`.

7.3.2.110 `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()`.

7.3.2.111 `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.

7.3.2.112 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__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>	
<code>__nu2</code>	

Definition at line 349 of file `sf_beta.tcc`.

References `__beta_inc()`.

7.3.2.113 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__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 378 of file `sf_beta.tcc`.

References `__beta_inc()`.

7.3.2.114 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__factorial (unsigned int __n)`

Return the factorial of the integer n.

The factorial is:

$$n! = 12... (n-1)n, 0! = 1$$

Definition at line 2422 of file `sf_gamma.tcc`.

References `_S_factorial_table`.

Referenced by `__double_factorial()`.

7.3.2.115 `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 1347 of file sf_polylog.tcc.

References `__polylog_exp()`.

7.3.2.116 `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 569 of file sf_mod_bessel.tcc.

7.3.2.117 `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 $\max(a, b) * 5 * eps$

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()`.

7.3.2.118 `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 $\text{Re}(w)$ is zero within $5 * \text{epsilon}$, `false` otherwise.

Definition at line 107 of file `sf_polylog.tcc`.

References `__fpequal()`.

7.3.2.119 `template<typename _Tp> bool std::__detail::__fpimag (const _Tp)`

Definition at line 117 of file `sf_polylog.tcc`.

7.3.2.120 `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 $\text{Im}(w)$ is zero within $5 * \text{epsilon}$, `false` otherwise.

Definition at line 84 of file `sf_polylog.tcc`.

References `__fpequal()`.

Referenced by `__polylog_exp_int_pos()`, and `__polylog_exp_real_pos()`.

7.3.2.121 `template<typename _Tp> bool std::__detail::__fpreal (const _Tp)`

Definition at line 94 of file `sf_polylog.tcc`.

7.3.2.122 `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

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

Definition at line 166 of file `sf_fresnel.tcc`.

References `__fresnel_cont_frac()`, and `__fresnel_series()`.

7.3.2.123 `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()`.

7.3.2.124 `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()`.

7.3.2.125 `template<typename _Tp> _Tp std::__detail::__gamma (_Tp __x)`

Return $\Gamma(x)$.

Parameters

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

Returns

The gamma function.

Definition at line 1918 of file `sf_gamma.tcc`.

References `__log_gamma()`.

Referenced by `__beta_gamma()`, and `__riemann_zeta()`.

7.3.2.126 `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()`.

7.3.2.127 `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()`.

7.3.2.128 `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()`.

7.3.2.129 `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()`.

7.3.2.130 `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()`.

7.3.2.131 `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()`.

7.3.2.132 `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()`.

7.3.2.133 `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`.

7.3.2.134 `template<typename _Tp> _Tp std::__detail::__gegenbauer_poly (unsigned int __n, _Tp __alpha, _Tp __x)`

Definition at line 44 of file `sf_gegenbauer.tcc`.

7.3.2.135 `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()`.

7.3.2.136 `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()`.

7.3.2.137 `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()`.

7.3.2.138 `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()`.

```
7.3.2.139 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()`.

```
7.3.2.140 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 `nu` appearing in Olver's uniform asymptotic expansions of the Hankel functions of the first and second kinds and their derivatives. The various functions of `z` and `nu` returned by `hankel_uniform_outer` are available for use in computing further terms in the expansions.

Definition at line 273 of file `sf_hankel.tcc`.

References `__airy()`, `__airy_arg()`, and `__hankel_params()`.

Referenced by `__hankel_uniform_olver()`.

7.3.2.141 `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()`.

7.3.2.142 `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()`.

7.3.2.143 `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 $s! = 1$
<code>__a</code>	The scale parameter $a > -1$

Definition at line 702 of file `sf_zeta.tcc`.

References `__hurwitz_zeta_euler_maclaurin()`.

Referenced by `__psi()`.

7.3.2.144 `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()`.

7.3.2.145 `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()`.

7.3.2.146 `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()`.

7.3.2.147 `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()`.

7.3.2.148 `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()`.

7.3.2.149 `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()`.

7.3.2.150 `template<typename _Tp> std::tuple<_Tp, _Tp, _Tp> std::__detail::__jacobi_sncndn (_Tp __k, _Tp __u)`

Return a tuple of the three primary Jacobi elliptic functions: $sn(k, u)$, $cn(k, u)$, $dn(k, u)$.

Definition at line 414 of file `sf_theta.tcc`.

7.3.2.151 `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()`.

7.3.2.152 `template<typename _Tp> _Tp std::__detail::__laguerre (unsigned int __n, _Tp __x)`

This routine returns the Laguerre polynomial of order n : $L_n(x)$.

The Laguerre polynomial is defined by:

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

Parameters

<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 312 of file `sf_laguerre.tcc`.

7.3.2.153 `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.

7.3.2.154 `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()`.

7.3.2.155 `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`.

7.3.2.156 `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()`.

7.3.2.157 `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()`.

7.3.2.158 `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`.

7.3.2.159 `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()`.

7.3.2.160 `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()`.

7.3.2.161 `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`.

7.3.2.162 `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, $\text{sf}[(n)_n = n!]$. 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.

7.3.2.163 `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()`.

7.3.2.164 `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()`.

7.3.2.165 `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()`.

7.3.2.166 `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()`.

7.3.2.167 `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`.

7.3.2.168 `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()`.

7.3.2.169 `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()`.

7.3.2.170 `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()`.

7.3.2.171 `template<typename _Tp> _Tp std::__detail::__poly_jacobi (unsigned int __n, _Tp __alpha, _Tp __beta, _Tp __x)`

Compute the Jacobi polynomial by recursion on x:

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

Definition at line 57 of file sf_jacobi.tcc.

References `__beta()`.

Referenced by `__poly_radial_jacobi()`.

7.3.2.172 `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:

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

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 240 of file sf_laguerre.tcc.

References `__poly_laguerre_hyperrg()`, `__poly_laguerre_large_n()`, and `__poly_laguerre_recursion()`.

7.3.2.173 `template<typename _Tpa, typename _Tp> _Tp std::__detail::__poly_laguerre_hyperrg (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.

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 125 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

7.3.2.174 `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 α for large n . Abramowitz & Stegun, 13.5.21.

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 70 of file `sf_laguerre.tcc`.

References `__log_gamma()`.

Referenced by `__poly_laguerre()`.

7.3.2.175 `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:

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

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 181 of file `sf_laguerre.tcc`.

Referenced by `__poly_laguerre()`.

7.3.2.176 `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()`.

7.3.2.177 `template<typename _Tp> _Tp std::__detail::__poly_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 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 polynomial. $l \geq 0$.
<code>__x</code>	The argument of the Legendre polynomial. $ x \leq 1$.

Definition at line 123 of file `sf_legendre.tcc`.

7.3.2.178 `template<typename _Tp> _Tp std::__detail::__poly_radial_jacobi (unsigned int __n, unsigned int __m, _Tp __rho)`

Definition at line 111 of file `sf_jacobi.tcc`.

References `__poly_jacobi()`.

Referenced by `__zernike()`, `__gnu_cxx::radpolyf()`, and `__gnu_cxx::radpolyf()`.

7.3.2.179 `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()`.

7.3.2.180 `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()`.

7.3.2.181 `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()`, and `__polylog()`.

7.3.2.182 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_asymp (const _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 $\text{Re}(w) \gg 1$

Don't check this against Mathematica 8. For real u the imaginary part of the polylog is given by $\text{Im}(Li_s(e^u)) = -\pi u^{s-1} / \Gamma(s)$ 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()`.

7.3.2.183 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_neg (const 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()`.

7.3.2.184 `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()`.

7.3.2.185 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_pos (const 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()`.

7.3.2.186 `template<typename _Tp> std::complex<_Tp> std::__detail::__polylog_exp_int_pos (const 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()`.

7.3.2.187 `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()`.

7.3.2.188 `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()`.

7.3.2.189 `template<typename _Tp, int __sigma> std::complex<_Tp> std::__detail::__polylog_exp_neg_even (unsigned int __n, std::complex<_Tp> __w)`

This function treats the cases of negative integer index s which are multiples of two.

In that case the sine occuring in the expansion occasionally takes on the value zero. We use that to provide an optimized series for $p = 2n$:

In the template parameter `sigma` we transport whether $p = 4k$ ($\sigma = 1$) or $p = 4k + 2$ ($\sigma = -1$)

$$Li_p(e^w) = \Gamma(1-p)(-w)^{p-1} - A_p(w) - \sigma * B_p(w)$$

with

$$A_p(w) = 2(2\pi)^{(p-1)}(-p)!/(2\pi)^{(p-1/2+p/2)} \cos((1-p)ArcTan(2\pi/w))$$

and

$$B_p(w) = -2(2\pi i)^{(p-1)} \sum_{k=0}^{\infty} \Gamma(2+2k-p)/(2k+1)! (-1)^k (w/2\pi)^{(2k+1)} (\zeta(2+2k-p) - \tau p1)$$

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)

$$Li_p(e^w) = \Gamma(1-p)(-w)^{p-1} - \sigma(2\pi)^p/p! \sum_{k=0}^{\infty} \Gamma(2+2k-p)/(2k+1)! (-1)^k (w/2\pi)^{(2k+1)} \zeta(2+2k-p)$$

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()`.

7.3.2.190 `template<typename _Tp, int __sigma> std::complex<_Tp> std::__detail::__polylog_exp_neg_odd (unsigned int __n, std::complex<_Tp> __w)`

This function treats the cases of negative integer index s which are odd.

In that case the sine occurring in the expansion occasionally vanishes. We use that to provide an optimized series for $p = 1+2k$: In the template parameter sigma we transport whether $p = 1+4k$ ($\sigma = 1$) or $p = 3+4k$ ($\sigma = -1$)

$$Li_p(e^w) = \Gamma(1-p) * (-w)^{p-1} + \sigma * A_p(w) - \sigma * B_p(w)$$

with

$$A_p(w) = 2(2\pi)^{(p-1)} * \Gamma(1-p) (1 + w^2/(4\pi^2))^{-1/2+p/2} \cos((1-p)ArcTan(2\pi i/w))$$

and

$$B_p(w) = 2(2\pi i)^{(p-1)} * \sum_{k=0}^{\infty} \Gamma(1+2k-p)/(2k)! (-w^2/4/\pi^2)^k (\zeta(1+2k-p) - \tau p1)$$

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)

$$Li_p(e^w) = Gamma(1-p) * (-w)^{p-1} - \sigma * 2 * (2\pi i)^{(p-1)} * \sum_{k=0}^{\infty} \Gamma(1+2k-p)/(2k)! (-1)^k (w/2/\pi)^{(2k)} \zeta(1+2k-p)$$

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()`.

7.3.2.191 `template<typename _PowTp, typename _Tp> _Tp std::__detail::__polylog_exp_negative_real_part (_PowTp __s, _Tp __w)`

Theoretical convergence for $\text{Re}(w) < 0$.

Seems to beat the other expansions for $\text{Re}(w) < -\pi/2 - \pi/5$. Note that this is an implementation of the basic series:

$$Li_s(e^z) = \sum_{k=1} e^{(k * z) * k^{(s-1)}}$$

Parameters

<code>__s</code>	is an arbitrary type, integral or float.
<code>__w</code>	something with a negative real part.

Returns

the value of the polylogarithm.

Definition at line 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()`.

7.3.2.192 `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 <code>s</code> .
<code>__w</code>	the argument <code>w</code> .

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()`.

7.3.2.193 `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()`.

7.3.2.194 `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()`.

7.3.2.195 `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_negative_real_part()`.

Referenced by `__polylog_exp()`.

7.3.2.196 `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()`.

7.3.2.197 `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()`.

7.3.2.198 `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()`.

7.3.2.199 `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()`.

7.3.2.200 `template<typename _Tp> _Tp std::__detail::__psi (unsigned int __n, _Tp __x)`

Return the polygamma function $\psi^{(n)}(x)$.

The polygamma function is related to the Hurwitz zeta function:

$$\psi^{(n)}(x) = (-1)^{n+1} n! \zeta(n+1, x)$$

Definition at line 2395 of file sf_gamma.tcc.

References `__hurwitz_zeta()`, `__log_gamma()`, and `__psi()`.

7.3.2.201 `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()`.

7.3.2.202 `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.

7.3.2.203 `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()`.

7.3.2.204 `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`.

7.3.2.205 `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`.

7.3.2.206 `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()`.

7.3.2.207 `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()`.

7.3.2.208 `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()`.

7.3.2.209 `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()`.

7.3.2.210 `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()`.

7.3.2.211 `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.

7.3.2.212 `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.

7.3.2.213 `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.

7.3.2.214 `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()`.

7.3.2.215 `template<typename _Tp> void std::__detail::__sincosint_asyp (_Tp __t, _Tp & _Si, _Tp & _Ci)`

This function computes the sine $Si(x)$ and cosine $Ci(x)$ integrals by asymptotic series summation for positive argument.

The asymptotic series is very good for $x > 50$.

Definition at line 163 of file `sf_trigint.tcc`.

Referenced by `__sincosint()`.

7.3.2.216 `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()`.

7.3.2.217 `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()`.

7.3.2.218 `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`.

7.3.2.219 `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`.

7.3.2.220 `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`.

7.3.2.221 `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()`.

7.3.2.222 `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 675 of file `sf_bessel.tcc`.

References `__sph_bessel_jn()`.

7.3.2.223 `template<typename _Tp> std::complex<_Tp> std::__detail::__sph_bessel (unsigned int __n, std::complex<_Tp> __z)`

Return the complex spherical Bessel function.

Parameters

<code>in</code>	<code>__n</code>	The order for which the spherical Bessel function is evaluated.
<code>in</code>	<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()`.

7.3.2.224 `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 445 of file `sf_mod_bessel.tcc`.

References `__cyl_bessel_ik()`.

7.3.2.225 `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 640 of file `sf_bessel.tcc`.

References `__cyl_bessel_jn()`.

Referenced by `__sph_bessel()`, `__sph_hankel_1()`, `__sph_hankel_2()`, and `__sph_neumann()`.

7.3.2.226 `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()`.

7.3.2.227 `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 744 of file `sf_bessel.tcc`.

References `__sph_bessel_jn()`.

7.3.2.228 `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

<code>in</code>	<code>__n</code>	The order for which the spherical Hankel function of the first kind is evaluated.
<code>in</code>	<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()`.

7.3.2.229 `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 777 of file `sf_bessel.tcc`.

References `__sph_bessel_jn()`.

7.3.2.230 `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()`.

7.3.2.231 `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()`.

7.3.2.232 `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()`.

7.3.2.233 `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 712 of file `sf_bessel.tcc`.

References `__sph_bessel_jn()`.

7.3.2.234 `template<typename _Tp> std::complex<_Tp> std::detail::__sph_neumann (unsigned int __n, std::complex<_Tp> __z)`

Return the complex spherical Neumann function.

Parameters

<code>in</code>	<code>__n</code>	The order for which the spherical Neumann function is evaluated.
<code>in</code>	<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()`.

7.3.2.235 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__students_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()`.

7.3.2.236 `template<typename _Tp > _GLIBCXX14_CONSTEXPR _Tp std::__detail::__students_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()`.

7.3.2.237 `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()`.

7.3.2.238 `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()`.

7.3.2.239 `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()`.

7.3.2.240 `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()`.

7.3.2.241 `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()`.

7.3.2.242 `template<typename _Tp> _Tp std::__detail::__theta_3_asyp (_Tp __nu, _Tp __x)`

Compute and return the θ_3 function by asymptotic series expansion.

Definition at line 128 of file sf_theta.tcc.

Referenced by `__theta_3()`.

7.3.2.243 `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()`.

7.3.2.244 `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()`.

7.3.2.245 `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()`.

7.3.2.246 `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()`.

7.3.2.247 `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()`.

7.3.2.248 `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()`.

7.3.2.249 `template<typename _Tp> __gnu_cxx::__promote_num_t<_Tp> std::__detail::__zernike (unsigned int __n, int __m, _Tp __rho, _Tp __phi)`

Definition at line 133 of file sf_jacobi.tcc.

References `__poly_radial_jacobi()`.

7.3.2.250 `template<typename _Tp> _Tp std::__detail::__znorm1 (_Tp __x)`

Definition at line 58 of file sf_owens_t.tcc.

Referenced by `__owens_t()`.

7.3.2.251 `template<typename _Tp> _Tp std::__detail::__znorm2 (_Tp __x)`

Definition at line 47 of file sf_owens_t.tcc.

Referenced by `__owens_t()`.

7.3.2.252 `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()`.

7.3.3 Variable Documentation

7.3.3.1 `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.

7.3.3.2 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().

7.3.3.3 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().

7.3.3.4 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().

7.3.3.5 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().

7.3.3.6 template<typename _Tp> constexpr std::size_t std::__detail::_S_num_double_factorials = 0

Definition at line 259 of file sf_gamma.tcc.

7.3.3.7 template<> constexpr std::size_t std::__detail::_S_num_double_factorials< double > = 301

Definition at line 264 of file sf_gamma.tcc.

7.3.3.8 template<> constexpr std::size_t std::__detail::_S_num_double_factorials< float > = 57

Definition at line 262 of file sf_gamma.tcc.

7.3.3.9 template<> constexpr std::size_t std::__detail::_S_num_double_factorials< long double > = 301

Definition at line 266 of file sf_gamma.tcc.

7.3.3.10 template<typename _Tp> constexpr std::size_t std::__detail::_S_num_factorials = 0

Definition at line 69 of file sf_gamma.tcc.

7.3.3.11 `template<> constexpr std::size_t std::__detail::_S_num_factorials< double > = 171`

Definition at line 74 of file `sf_gamma.tcc`.

7.3.3.12 `template<> constexpr std::size_t std::__detail::_S_num_factorials< float > = 35`

Definition at line 72 of file `sf_gamma.tcc`.

7.3.3.13 `template<> constexpr std::size_t std::__detail::_S_num_factorials< long double > = 171`

Definition at line 76 of file `sf_gamma.tcc`.

7.3.3.14 `template<typename _Tp > constexpr std::size_t std::__detail::_S_num_neg_double_factorials = 0`

Definition at line 579 of file `sf_gamma.tcc`.

7.3.3.15 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< double > = 150`

Definition at line 584 of file `sf_gamma.tcc`.

7.3.3.16 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< float > = 27`

Definition at line 582 of file `sf_gamma.tcc`.

7.3.3.17 `template<> constexpr std::size_t std::__detail::_S_num_neg_double_factorials< long double > = 999`

Definition at line 586 of file `sf_gamma.tcc`.

7.3.3.18 `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()`.

7.3.3.19 `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 8

Class Documentation

8.1 `std::__detail::_Factorial_table<_Tp>` Struct Template Reference

Public Attributes

- `_Tp __factorial`
- `_Tp __log_factorial`
- `unsigned int __n`

8.1.1 Detailed Description

```
template<typename _Tp>struct std::__detail::_Factorial_table<_Tp>
```

Definition at line 61 of file `sf_gamma.tcc`.

8.1.2 Member Data Documentation

8.1.2.1 `template<typename _Tp>_Tp std::__detail::_Factorial_table<_Tp>::__factorial`

Definition at line 64 of file `sf_gamma.tcc`.

8.1.2.2 `template<typename _Tp>_Tp std::__detail::_Factorial_table<_Tp>::__log_factorial`

Definition at line 65 of file `sf_gamma.tcc`.

8.1.2.3 `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`

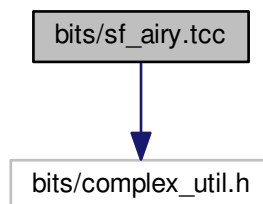
Chapter 9

File Documentation

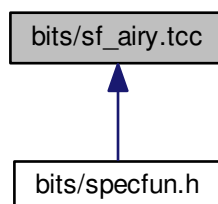
9.1 bits/sf_airy.tcc File Reference

```
#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.

9.1.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

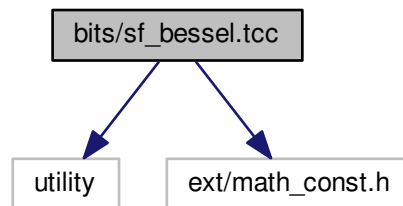
9.1.2 Macro Definition Documentation

9.1.2.1 `#define _GLIBCXX_BITS_SF_AIRY_TCC 1`

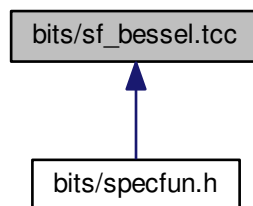
Definition at line 31 of file `sf_airy.tcc`.

9.2 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 .

9.2.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.2.2 Macro Definition Documentation

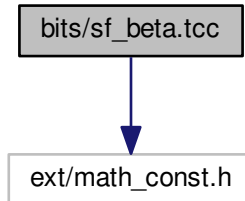
9.2.2.1 `#define _GLIBCXX_BITS_SF_BESSEL_TCC 1`

Definition at line 47 of file `sf_bessel.tcc`.

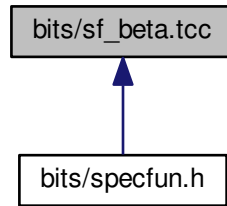
9.3 bits/sf_beta.tcc File Reference

```
#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::__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::__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::__students_t_cdf (_Tp __t, unsigned int __nu)`
Return the Students T probability function.
- `template<typename _Tp >`
`_GLIBCXX14_CONSTEXPR _Tp std::__detail::__students_t_cdfc (_Tp __t, unsigned int __nu)`
Return the complement of the Students T probability function.

9.3.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.3.2 Macro Definition Documentation

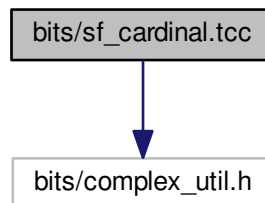
9.3.2.1 `#define _GLIBCXX_BITS_SF_BETA_TCC 1`

Definition at line 49 of file `sf_beta.tcc`.

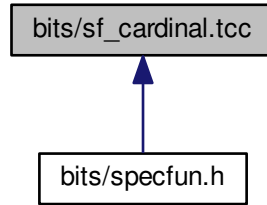
9.4 bits/sf_cardinal.tcc File Reference

```
#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

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

9.4.1 Macro Definition Documentation

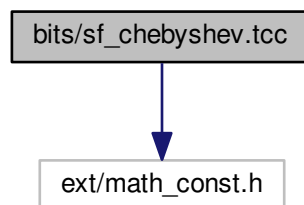
9.4.1.1 #define _GLIBCXX_BITS_SF_CARDINAL_TCC 1

Definition at line 30 of file sf_cardinal.tcc.

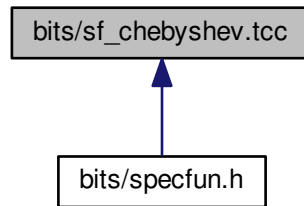
9.5 bits/sf_chebyshev.tcc File Reference

`#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 \(unsigned int __n, _Tp __x\)](#)

9.5.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.5.2 Macro Definition Documentation

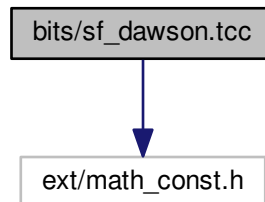
9.5.2.1 [#define _GLIBCXX_SF_CHEBYSHEV_TCC 1](#)

Definition at line 31 of file `sf_chebyshev.tcc`.

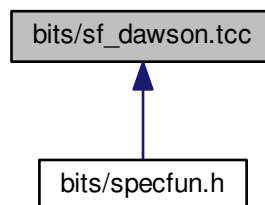
9.6 bits/sf_dawson.tcc File Reference

```
#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_const_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.

9.6.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.6.2 Macro Definition Documentation

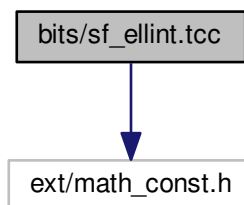
9.6.2.1 `#define _GLIBCXX_SF_DAWSON_TCC 1`

Definition at line 31 of file `sf_dawson.tcc`.

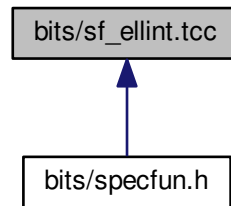
9.7 bits/sf_ellint.tcc File Reference

```
#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)`

9.7.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.7.2 Macro Definition Documentation

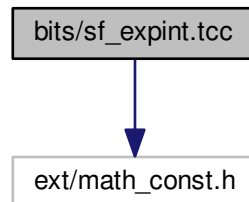
9.7.2.1 `#define _GLIBCXX_BITS_SF_ELLINT_TCC 1`

Definition at line 47 of file `sf_ellint.tcc`.

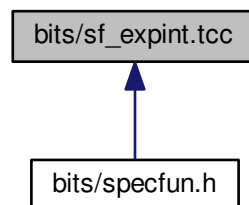
9.8 bits/sf_expint.tcc File Reference

```
#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 $li(x)$.

9.8.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.8.2 Macro Definition Documentation

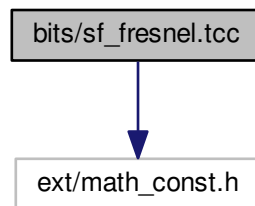
9.8.2.1 `#define _GLIBCXX_BITS_SF_EXPINT_TCC 1`

Definition at line 47 of file `sf_expint.tcc`.

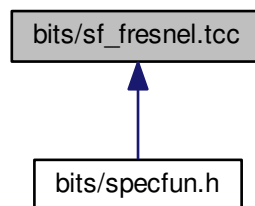
9.9 bits/sf_fresnel.tcc File Reference

```
#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.

9.9.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.9.2 Macro Definition Documentation

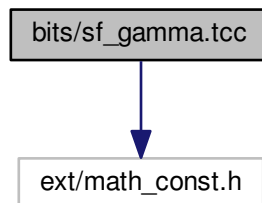
9.9.2.1 [#define _GLIBCXX_SF_FRESNEL_TCC](#) 1

Definition at line 31 of file `sf_fresnel.tcc`.

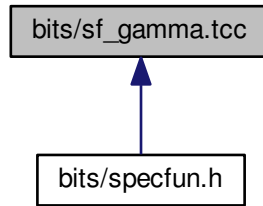
9.10 bits/sf_gamma.tcc File Reference

```
#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, $(n)_n = n!$. Thus this function returns

$$\ln[(a)_n] = \Gamma(a+1) - \Gamma(a-n+1), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

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

, and others.

- `template<typename _Tp >`
`_Tp std::__detail::__log_pochhammer_u (_Tp __a, _Tp __n)`

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

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(a)$$

Thus this function returns

$$\ln[(a)_n] = \Gamma(a+n) - \Gamma(a), \ln[(a)_0] = 0$$

Many notations exist:

$$a^{\overline{n}}$$

,

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

, and others.

- `template<typename _Tp >`
`_Tp std::__detail::__pochhammer_l (_Tp __a, _Tp __n)`

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

$$(a)_n = \prod_{k=0}^{n-1} (a-k), (a)_0 = 1 = \Gamma(a+1)/\Gamma(a-n+1)$$

In particular, $(n)_n = n!$.

- `template<typename _Tp >`
`_Tp std::__detail::__pochhammer_u (_Tp __a, _Tp __n)`

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

$$(a)_n = \prod_{k=0}^{n-1} (a+k), (a)_0 = 1 = \Gamma(a+n)/\Gamma(a)$$

Many notations exist:

$$a^{\overline{n}}$$

,

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

, and others.

- `template<typename _Tp >`
`_Tp std::__detail::__psi (_Tp __x)`

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

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

For negative argument the reflection formula is used:

$$\psi(x) = \psi(1-x) - \pi \cot(\pi x)$$

.

- `template<typename _Tp >`
`_Tp std::__detail::__psi (unsigned int __n, _Tp __x)`

Return the polygamma function $\psi^{(n)}(x)$.

- `template<typename _Tp >`
`_Tp std::__detail::__psi_asymp (_Tp __x)`

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

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

- `template<typename _Tp >`
`_Tp std::__detail::__psi_series (_Tp __x)`

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

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

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`

9.10.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.10.2 Macro Definition Documentation

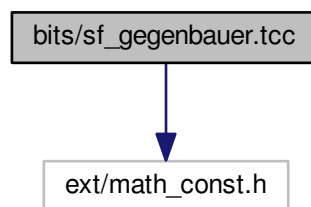
9.10.2.1 `#define _GLIBCXX_BITS_SF_GAMMA_TCC 1`

Definition at line 49 of file `sf_gamma.tcc`.

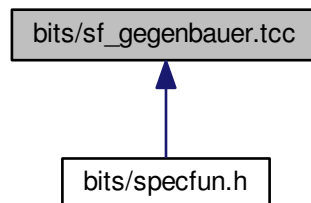
9.11 `bits/sf_gegenbauer.tcc` File Reference

```
#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)

9.11.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

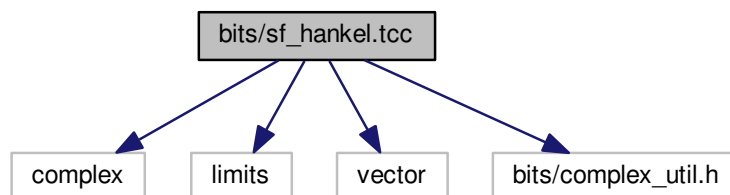
9.11.2 Macro Definition Documentation

9.11.2.1 `#define _GLIBCXX_SF_GEGENBAUER_TCC 1`

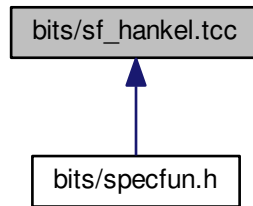
Definition at line 31 of file `sf_gegenbauer.tcc`.

9.12 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.

9.12.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

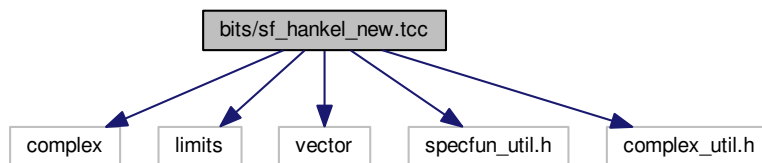
9.12.2 Macro Definition Documentation

9.12.2.1 `#define _GLIBCXX_BITS_SF_HANKEL_TCC 1`

Definition at line 31 of file `sf_hankel.tcc`.

9.13 `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`

9.13.1 Macro Definition Documentation

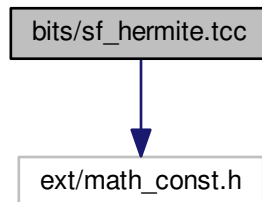
9.13.1.1 `#define _GLIBCXX_BITS_SF_HANKEL_NEW_TCC 1`

Definition at line 31 of file `sf_hankel_new.tcc`.

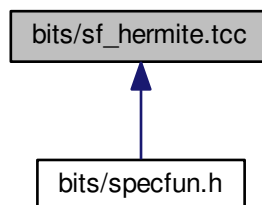
9.14 bits/sf_hermite.tcc File Reference

```
#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 .

9.14.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

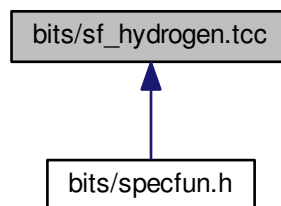
9.14.2 Macro Definition Documentation

9.14.2.1 `#define _GLIBCXX_BITS_SF_HERMITE_TCC 1`

Definition at line 42 of file `sf_hermite.tcc`.

9.15 `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)

9.15.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

9.15.2 Macro Definition Documentation

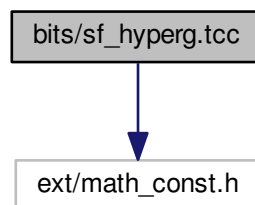
9.15.2.1 `#define _GLIBCXX_BITS_SF_HYDROGEN_TCC 1`

Definition at line 31 of file `sf_hydrogen.tcc`.

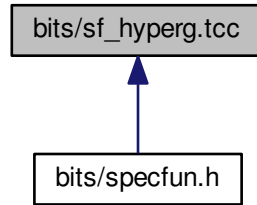
9.16 bits/sf_hyperg.tcc File Reference

```
#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.

9.16.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.16.2 Macro Definition Documentation

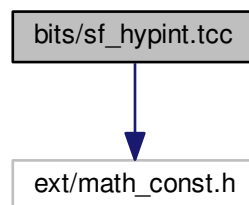
9.16.2.1 `#define _GLIBCXX_BITS_SF_HYPERG_TCC 1`

Definition at line 44 of file `sf_hyperg.tcc`.

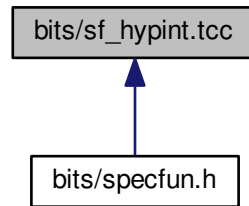
9.17 bits/sf_hypint.tcc File Reference

`#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.

9.17.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.17.2 Macro Definition Documentation

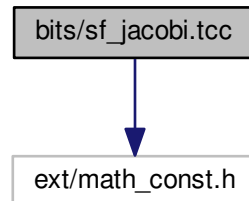
9.17.2.1 [#define _GLIBCXX_SF_HYPINT_TCC 1](#)

Definition at line 31 of file `sf_hypint.tcc`.

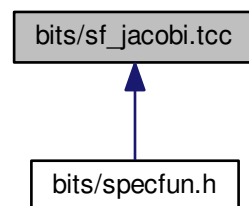
9.18 bits/sf_jacobi.tcc File Reference

```
#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)

9.18.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.18.2 Macro Definition Documentation

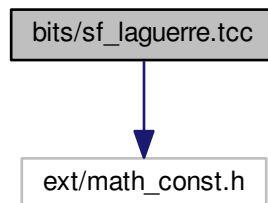
9.18.2.1 `#define _GLIBCXX_SF_JACOBI_TCC 1`

Definition at line 31 of file `sf_jacobi.tcc`.

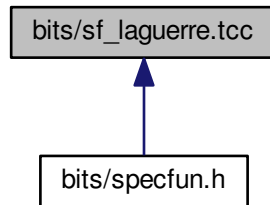
9.19 bits/sf_laguerre.tcc File Reference

```
#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 α 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.

9.19.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.19.2 Macro Definition Documentation

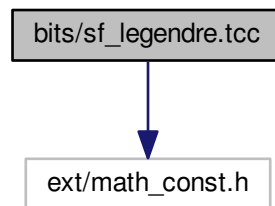
9.19.2.1 `#define _GLIBCXX_BITS_SF_LAGUERRE_TCC 1`

Definition at line 44 of file `sf_laguerre.tcc`.

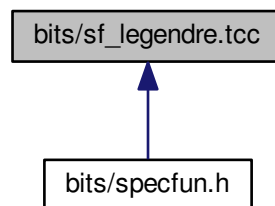
9.20 `bits/sf_legendre.tcc` File Reference

```
#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::__poly_legendre_p (unsigned int __l, _Tp __x)`
Return the Legendre polynomial by upward recursion on order l .
- `template<typename _Tp >`
`_Tp std::__detail::__poly_legendre_q (unsigned int __l, _Tp __x)`
Return the Legendre function of the second kind by upward recursion on order l .
- `template<typename _Tp >`
`std::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.

9.20.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.20.2 Macro Definition Documentation

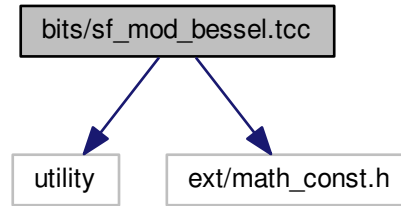
9.20.2.1 [#define _GLIBCXX_BITS_SF_LEGENDRE_TCC 1](#)

Definition at line 47 of file `sf_legendre.tcc`.

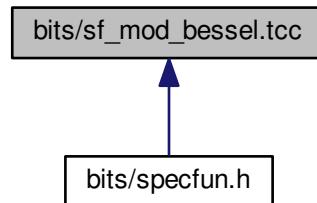
9.21 bits/sf_mod_bessel.tcc File Reference

```
#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.

9.21.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

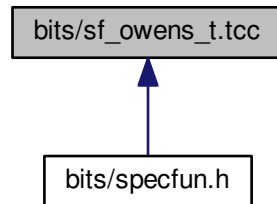
9.21.2 Macro Definition Documentation

9.21.2.1 `#define GLIBCXX_BITS_SF_MOD_BESSEL_TCC 1`

Definition at line 47 of file `sf_mod_bessel.tcc`.

9.22 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 \(_Tp __x\)](#)

9.22.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly.

9.22.2 Macro Definition Documentation

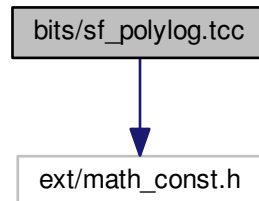
9.22.2.1 [#define _GLIBCXX_BITS_SF_OWENS_T_TCC 1](#)

Definition at line 31 of file `sf_owens_t.tcc`.

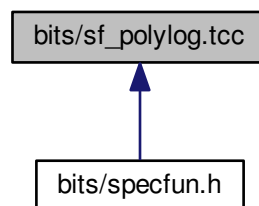
9.23 bits/sf_polylog.tcc File Reference

```
#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 >`
`_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 (const _Tp __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_int_neg (const 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 (const unsigned int __s, std::complex< _Tp > __w)`
- `template<typename _Tp >`
`std::complex< _Tp > std::__detail::__polylog_exp_int_pos (const 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)`

9.23.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.23.2 Macro Definition Documentation

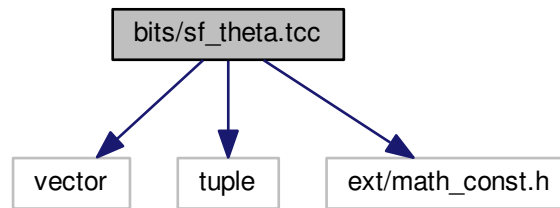
9.23.2.1 `#define _GLIBCXX_BITS_SF_POLYLOG_TCC 1`

Definition at line 41 of file `sf_polylog.tcc`.

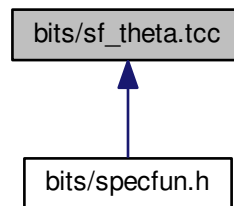
9.24 bits/sf_theta.tcc File Reference

```
#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)`

9.24.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

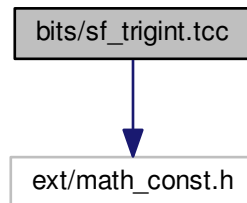
9.24.2 Macro Definition Documentation

9.24.2.1 `#define _GLIBCXX_SF_THETA_TCC 1`

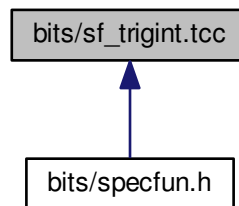
Definition at line 31 of file `sf_theta.tcc`.

9.25 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.

9.25.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.25.2 Macro Definition Documentation

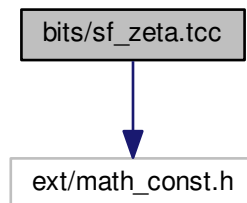
9.25.2.1 `#define _GLIBCXX_SF_TRIGINT_TCC 1`

Definition at line 31 of file `sf_trigint.tcc`.

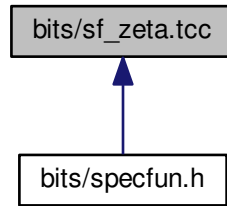
9.26 bits/sf_zeta.tcc File Reference

```
#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]`

9.26.1 Detailed Description

This is an internal header file, included by other library headers. Do not attempt to use it directly. Instead, include `<cmath>`.

9.26.2 Macro Definition Documentation

9.26.2.1 `#define _GLIBCXX_BITS_SF_ZETA_TCC 1`

Definition at line 46 of file `sf_zeta.tcc`.

9.27 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::bincoeff](#) (unsigned int __n, unsigned int __k)
- long double [__gnu_cxx::bincoefl](#) (unsigned int __n, unsigned int __k)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_t](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_tf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_tl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_u](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_uf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_ul](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_v](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_vf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_vl](#) (unsigned int __n, long double __x)
- template<typename _Tp >
[__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::chebyshev_w](#) (unsigned int __n, _Tp __x)
- float [__gnu_cxx::chebyshev_wf](#) (unsigned int __n, float __x)
- long double [__gnu_cxx::chebyshev_wl](#) (unsigned int __n, long double __x)

- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen` (unsigned int __m, _Tp __w)
- `template<typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tp > > __gnu_cxx::clausen` (unsigned int __m, std::complex< _Tp > __w)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_c` (unsigned int __m, _Tp __w)
- `float __gnu_cxx::clausen_cf` (unsigned int __m, float __w)
- `long double __gnu_cxx::clausen_cl` (unsigned int __m, long double __w)
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::clausen_s` (unsigned int __m, _Tp __w)
- `float __gnu_cxx::clausen_sf` (unsigned int __m, float __w)
- `long double __gnu_cxx::clausen_sl` (unsigned int __m, long double __w)
- `float __gnu_cxx::clausenf` (unsigned int __m, float __w)
- `std::complex< float > __gnu_cxx::clausenf` (unsigned int __m, std::complex< float > __w)
- `long double __gnu_cxx::clausenl` (unsigned int __m, long double __w)
- `std::complex< long double > __gnu_cxx::clausenl` (unsigned int __m, std::complex< long double > __w)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_1` (_Tp __k)
- `float std::comp_ellint_1f` (float __k)
- `long double std::comp_ellint_1l` (long double __k)
- `template<typename _Tp >`
`__gnu_cxx::__promote< _Tp >::__type std::comp_ellint_2` (_Tp __k)
- `float std::comp_ellint_2f` (float __k)
- `long double std::comp_ellint_2l` (long double __k)
- `template<typename _Tp, typename _Tpn >`
`__gnu_cxx::__promote_2< _Tp, _Tpn >::__type std::comp_ellint_3` (_Tp __k, _Tpn __nu)
- `float std::comp_ellint_3f` (float __k, float __nu)
- Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for float modulus k.*
- `long double std::comp_ellint_3l` (long double __k, long double __nu)
- Return the complete elliptic integral of the third kind $\Pi(k, \nu)$ for long double modulus k.*
- `template<typename _Tk >`
`__gnu_cxx::__promote_num_t< _Tk > __gnu_cxx::comp_ellint_d` (_Tk __k)
- `float __gnu_cxx::comp_ellint_df` (float __k)
- `long double __gnu_cxx::comp_ellint_dl` (long double __k)
- `float __gnu_cxx::comp_ellint_rf` (float __x, float __y)
- `long double __gnu_cxx::comp_ellint_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< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __nu, std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_1f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_1l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (_Tpnu __nu, _Tp __z)`
- `template<typename _Tpnu, typename _Tp >`
`std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __gnu_cxx::cyl_hankel_2 (std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __nu, std::complex< __gnu_cxx::__promote_num_t< _Tpnu, _Tp > > __x)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (float __nu, float __z)`
- `std::complex< float > __gnu_cxx::cyl_hankel_2f (std::complex< float > __nu, std::complex< float > __x)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (long double __nu, long double __z)`
- `std::complex< long double > __gnu_cxx::cyl_hankel_2l (std::complex< long double > __nu, std::complex< long double > __x)`
- `template<typename _Tpnu, typename _Tp >`
`__gnu_cxx::__promote_2< _Tpnu, _Tp >::type std::cyl_neumann (_Tpnu __nu, _Tp __x)`
- `float std::cyl_neumannf (float __nu, float __x)`
- `long double std::cyl_neumannl (long double __nu, long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::dawson (_Tp __x)`
- `float __gnu_cxx::dawsonf (float __x)`
- `long double __gnu_cxx::dawsonl (long double __x)`
- `template<typename _Tp >`
`__gnu_cxx::__promote_num_t< _Tp > __gnu_cxx::digamma (_Tp __z)`

- float `__gnu_cxx::digammaf` (float `__z`)
- long double `__gnu_cxx::digammal` (long double `__z`)
- template<typename `_Tp` >
`__gnu_cxx::__promote_num_t<_Tp> __gnu_cxx::dilog` (`_Tp` `__x`)
- float `__gnu_cxx::dilogf` (float `__x`)
- long double `__gnu_cxx::dilogl` (long double `__x`)
- template<typename `_Tp` >
`_Tp __gnu_cxx::dirichlet_beta` (`_Tp` `__x`)
- float `__gnu_cxx::dirichlet_betaf` (float `__x`)
- long double `__gnu_cxx::dirichlet_betall` (long double `__x`)
- template<typename `_Tp` >
`_Tp __gnu_cxx::dirichlet_eta` (`_Tp` `__x`)
- float `__gnu_cxx::dirichlet_etaf` (float `__x`)
- long double `__gnu_cxx::dirichlet_etaall` (long double `__x`)
- 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)`
- `float __gnu_cxx::hurwitz_zetaf (float __s, float __a)`
- `long double __gnu_cxx::hurwitz_zetal (long double __s, long double __a)`
- `template<typename _Tpa, typename _Tpb, typename _Tpc, typename _Tp >`
`__gnu_cxx::__promote_4< _Tpa, _Tpb, _Tpc, _Tp >::__type __gnu_cxx::hyperg (_Tpa __a, _Tpb __b, _Tpc __c, _Tp __x)`
- `float __gnu_cxx::hypergf (float __a, float __b, float __c, float __x)`
- `long double __gnu_cxx::hypergl (long double __a, long double __b, long double __c, long double __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibeta (_Ta __a, _Tb __b, _Tp __x)`
- `template<typename _Ta, typename _Tb, typename _Tp >`
`__gnu_cxx::__promote_num_t< _Ta, _Tb, _Tp > __gnu_cxx::ibetac (_Ta __a, _Tb __b, _Tp __x)`
- `float __gnu_cxx::ibetacf (float __a, float __b, float __x)`
- `long double __gnu_cxx::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 >
 std::complex< [__gnu_cxx::__promote_num_t<_Tp > > __gnu_cxx::polylog](#) (_Tp __s, std::complex< _Tp > __w)
- std::complex< float > [__gnu_cxx::polylogf](#) (float __s, std::complex< float > __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)

9.27.1 Detailed Description

This is an internal header file, included by other library headers. You should not attempt to use it directly. Instead, include `<cmath>`.

9.27.2 Macro Definition Documentation

9.27.2.1 `#define __cpp_lib_math_special_functions 201603L`

Definition at line 39 of file `specfun.h`.

9.27.2.2 `#define __STDCPP_MATH_SPEC_FUNCS__ 201003L`

Definition at line 37 of file `specfun.h`.

Index

`_GLIBCXX_BITS_SF_AIRY_TCC`
 `sf_airy.tcc`, [207](#)
`_GLIBCXX_BITS_SF_BESSEL_TCC`
 `sf_bessel.tcc`, [209](#)
`_GLIBCXX_BITS_SF_BETA_TCC`
 `sf_beta.tcc`, [211](#)
`_GLIBCXX_BITS_SF_CARDINAL_TCC`
 `sf_cardinal.tcc`, [213](#)
`_GLIBCXX_BITS_SF_ELLINT_TCC`
 `sf_ellint.tcc`, [218](#)
`_GLIBCXX_BITS_SF_EXPINT_TCC`
 `sf_expint.tcc`, [221](#)
`_GLIBCXX_BITS_SF_GAMMA_TCC`
 `sf_gamma.tcc`, [228](#)
`_GLIBCXX_BITS_SF_HANKEL_NEW_TCC`
 `sf_hankel_new.tcc`, [232](#)
`_GLIBCXX_BITS_SF_HANKEL_TCC`
 `sf_hankel.tcc`, [232](#)
`_GLIBCXX_BITS_SF_HERMITE_TCC`
 `sf_hermite.tcc`, [234](#)
`_GLIBCXX_BITS_SF_HYDROGEN_TCC`
 `sf_hydrogen.tcc`, [235](#)
`_GLIBCXX_BITS_SF_HYPERG_TCC`
 `sf_hyperg.tcc`, [237](#)
`_GLIBCXX_BITS_SF_LAGUERRE_TCC`
 `sf_laguerre.tcc`, [242](#)
`_GLIBCXX_BITS_SF_LEGENDRE_TCC`
 `sf_legendre.tcc`, [243](#)
`_GLIBCXX_BITS_SF_MOD_BESSEL_TCC`
 `sf_mod_bessel.tcc`, [245](#)
`_GLIBCXX_BITS_SF_OWENS_T_TCC`
 `sf_owens_t.tcc`, [246](#)
`_GLIBCXX_BITS_SF_POLYLOG_TCC`
 `sf_polylog.tcc`, [249](#)
`_GLIBCXX_BITS_SF_ZETA_TCC`
 `sf_zeta.tcc`, [255](#)
`_GLIBCXX_JACOBI_CN`
 Extended Mathematical Special Functions, [19](#)
`_GLIBCXX_JACOBI_DN`
 Extended Mathematical Special Functions, [19](#)
`_GLIBCXX_JACOBI_SN`
 Extended Mathematical Special Functions, [19](#)
`_GLIBCXX_SF_CHEBYSHEV_TCC`
 `sf_chebyshev.tcc`, [214](#)
`_GLIBCXX_SF_DAWSON_TCC`
 `sf_dawson.tcc`, [216](#)
`_GLIBCXX_SF_FRESNEL_TCC`
 `sf_fresnel.tcc`, [222](#)
`_GLIBCXX_SF_GEGENBAUER_TCC`
 `sf_gegenbauer.tcc`, [229](#)
`_GLIBCXX_SF_HYPINT_TCC`
 `sf_hypint.tcc`, [238](#)
`_GLIBCXX_SF_JACOBI_TCC`
 `sf_jacobi.tcc`, [240](#)
`_GLIBCXX_SF_THETA_TCC`
 `sf_theta.tcc`, [251](#)
`_GLIBCXX_SF_TRIGINT_TCC`
 `sf_trigint.tcc`, [253](#)
`_Num_Euler_Maclaurin_zeta`
 `std::__detail`, [200](#)
`_S_Euler_Maclaurin_zeta`
 `std::__detail`, [201](#)
`_S_double_factorial_table`
 `std::__detail`, [201](#)
`_S_factorial_table`
 `std::__detail`, [201](#)
`_S_neg_double_factorial_table`
 `std::__detail`, [201](#)
`_S_num_double_factorials`
 `std::__detail`, [201](#)
`_S_num_double_factorials< double >`
 `std::__detail`, [201](#)
`_S_num_double_factorials< float >`
 `std::__detail`, [201](#)
`_S_num_double_factorials< long double >`
 `std::__detail`, [201](#)
`_S_num_factorials`
 `std::__detail`, [201](#)
`_S_num_factorials< double >`
 `std::__detail`, [201](#)
`_S_num_factorials< float >`
 `std::__detail`, [202](#)
`_S_num_factorials< long double >`
 `std::__detail`, [202](#)
`_S_num_neg_double_factorials`
 `std::__detail`, [202](#)
`_S_num_neg_double_factorials< double >`
 `std::__detail`, [202](#)
`_S_num_neg_double_factorials< float >`
 `std::__detail`, [202](#)

- `__S_num_neg_double_factorials< long double >`
 - `std::__detail`, [202](#)
- `__S_num_zetam1`
 - `std::__detail`, [202](#)
- `__S_zetam1`
 - `std::__detail`, [202](#)
- `__STDCPP_MATH_SPEC_FUNCS__`
 - `specfun.h`, [266](#)
- `__airy`
 - `std::__detail`, [112](#), [113](#)
- `__airy_ai`
 - `std::__detail`, [115](#)
- `__airy_arg`
 - `std::__detail`, [115](#)
- `__airy_asymp_absarg_ge_pio3`
 - `std::__detail`, [115](#)
- `__airy_asymp_absarg_lt_pio3`
 - `std::__detail`, [116](#)
- `__airy_bessel_i`
 - `std::__detail`, [116](#)
- `__airy_bessel_k`
 - `std::__detail`, [117](#)
- `__airy_bi`
 - `std::__detail`, [118](#)
- `__airy_hyperg_rational`
 - `std::__detail`, [118](#)
- `__assoc_laguerre`
 - `std::__detail`, [119](#)
- `__assoc_legendre_p`
 - `std::__detail`, [119](#)
- `__bernoulli`
 - `std::__detail`, [119](#)
- `__bernoulli_2n`
 - `std::__detail`, [120](#)
- `__bernoulli_series`
 - `std::__detail`, [120](#)
- `__beta`
 - `std::__detail`, [120](#)
- `__beta_gamma`
 - `std::__detail`, [121](#)
- `__beta_inc`
 - `std::__detail`, [121](#)
- `__beta_inc_cont_frac`
 - `std::__detail`, [122](#)
- `__beta_lgamma`
 - `std::__detail`, [122](#)
- `__beta_product`
 - `std::__detail`, [122](#)
- `__bincoef`
 - `std::__detail`, [123](#)
- `__binomial_cdf`
 - `std::__detail`, [123](#)
- `__binomial_cdfc`
 - `std::__detail`, [123](#)
- `__bose_einstein`
 - `std::__detail`, [124](#)
- `__chebyshev_recur`
 - `std::__detail`, [124](#)
- `__chebyshev_t`
 - `std::__detail`, [124](#)
- `__chebyshev_u`
 - `std::__detail`, [124](#)
- `__chebyshev_v`
 - `std::__detail`, [124](#)
- `__chebyshev_w`
 - `std::__detail`, [124](#)
- `__chi_squared_pdf`
 - `std::__detail`, [125](#)
- `__chi_squared_pdfc`
 - `std::__detail`, [125](#)
- `__chshint`
 - `std::__detail`, [125](#)
- `__chshint_cont_frac`
 - `std::__detail`, [125](#)
- `__chshint_series`
 - `std::__detail`, [126](#)
- `__clamp_0_m2pi`
 - `std::__detail`, [126](#)
- `__clamp_pi`
 - `std::__detail`, [126](#)
- `__clausen`
 - `std::__detail`, [126](#)
- `__clausen_c`
 - `std::__detail`, [127](#)
- `__clausen_s`
 - `std::__detail`, [127](#), [128](#)
- `__comp_ellint_1`
 - `std::__detail`, [128](#)
- `__comp_ellint_2`
 - `std::__detail`, [129](#)
- `__comp_ellint_3`
 - `std::__detail`, [129](#)
- `__comp_ellint_d`
 - `std::__detail`, [129](#)
- `__comp_ellint_rf`
 - `std::__detail`, [130](#)
- `__comp_ellint_rg`
 - `std::__detail`, [130](#)
- `__conf_hyperg`
 - `std::__detail`, [130](#)
- `__conf_hyperg_lim`
 - `std::__detail`, [130](#)
- `__conf_hyperg_lim_series`
 - `std::__detail`, [130](#)
- `__conf_hyperg_luke`
 - `std::__detail`, [131](#)
- `__conf_hyperg_series`
 - `std::__detail`, [131](#)

- `__coshint`
 - `std::__detail`, [131](#)
- `__cpp_lib_math_special_functions`
 - `specfun.h`, [266](#)
- `__cyl_bessel`
 - `std::__detail`, [132](#)
- `__cyl_bessel_i`
 - `std::__detail`, [132](#)
- `__cyl_bessel_ij_series`
 - `std::__detail`, [133](#)
- `__cyl_bessel_ik`
 - `std::__detail`, [133](#)
- `__cyl_bessel_ik_asymp`
 - `std::__detail`, [134](#)
- `__cyl_bessel_ik_steel`
 - `std::__detail`, [134](#)
- `__cyl_bessel_j`
 - `std::__detail`, [134](#)
- `__cyl_bessel_jn`
 - `std::__detail`, [135](#)
- `__cyl_bessel_jn_asymp`
 - `std::__detail`, [135](#)
- `__cyl_bessel_jn_steel`
 - `std::__detail`, [135](#)
- `__cyl_bessel_k`
 - `std::__detail`, [136](#)
- `__cyl_hankel_1`
 - `std::__detail`, [136](#), [137](#)
- `__cyl_hankel_2`
 - `std::__detail`, [137](#)
- `__cyl_neumann`
 - `std::__detail`, [138](#)
- `__cyl_neumann_n`
 - `std::__detail`, [138](#)
- `__dawson`
 - `std::__detail`, [138](#)
- `__dawson_const_frac`
 - `std::__detail`, [139](#)
- `__dawson_series`
 - `std::__detail`, [139](#)
- `__debye_region`
 - `std::__detail`, [139](#)
- `__dilog`
 - `std::__detail`, [139](#)
- `__dirichlet_beta`
 - `std::__detail`, [140](#)
- `__dirichlet_eta`
 - `std::__detail`, [140](#), [141](#)
- `__double_factorial`
 - `std::__detail`, [141](#)
- `__ellint_1`
 - `std::__detail`, [141](#)
- `__ellint_2`
 - `std::__detail`, [142](#)
- `__ellint_3`
 - `std::__detail`, [142](#)
- `__ellint_cel`
 - `std::__detail`, [142](#)
- `__ellint_d`
 - `std::__detail`, [143](#)
- `__ellint_el1`
 - `std::__detail`, [143](#)
- `__ellint_el2`
 - `std::__detail`, [143](#)
- `__ellint_el3`
 - `std::__detail`, [143](#)
- `__ellint_rc`
 - `std::__detail`, [143](#)
- `__ellint_rd`
 - `std::__detail`, [144](#)
- `__ellint_rf`
 - `std::__detail`, [144](#)
- `__ellint_rg`
 - `std::__detail`, [145](#)
- `__ellint_rj`
 - `std::__detail`, [145](#)
- `__ellnome`
 - `std::__detail`, [146](#)
- `__ellnome_k`
 - `std::__detail`, [146](#)
- `__ellnome_series`
 - `std::__detail`, [146](#)
- `__expint`
 - `std::__detail`, [146](#), [147](#)
- `__expint_E1`
 - `std::__detail`, [147](#)
- `__expint_E1_asymp`
 - `std::__detail`, [148](#)
- `__expint_E1_series`
 - `std::__detail`, [148](#)
- `__expint_Ei`
 - `std::__detail`, [149](#)
- `__expint_Ei_asymp`
 - `std::__detail`, [149](#)
- `__expint_Ei_series`
 - `std::__detail`, [149](#)
- `__expint_En_cont_frac`
 - `std::__detail`, [150](#)
- `__expint_En_recursion`
 - `std::__detail`, [150](#)
- `__expint_En_series`
 - `std::__detail`, [151](#)
- `__expint_asymp`
 - `std::__detail`, [147](#)
- `__expint_large_n`
 - `std::__detail`, [151](#)
- `__f_cdf`
 - `std::__detail`, [151](#)

- __f_cdfc
 - std::__detail, 152
- __factorial
 - std::__detail, 152
 - std::__detail::Factorial_table, 203
- __fermi_dirac
 - std::__detail, 152
- __fock_airy
 - std::__detail, 153
- __fpequal
 - std::__detail, 153
- __fpimag
 - std::__detail, 153, 155
- __fpreal
 - std::__detail, 155
- __fresnel
 - std::__detail, 155
- __fresnel_cont_frac
 - std::__detail, 156
- __fresnel_series
 - std::__detail, 156
- __gamma
 - std::__detail, 156
- __gamma_cont_frac
 - std::__detail, 156
- __gamma_l
 - std::__detail, 156
- __gamma_p
 - std::__detail, 157
- __gamma_q
 - std::__detail, 157
- __gamma_series
 - std::__detail, 157
- __gamma_temme
 - std::__detail, 157
- __gamma_u
 - std::__detail, 158
- __gauss
 - std::__detail, 158
- __gegenbauer_poly
 - std::__detail, 158
- __gnu_cxx, 85
- __hankel
 - std::__detail, 158
- __hankel_debye
 - std::__detail, 159
- __hankel_params
 - std::__detail, 159
- __hankel_uniform
 - std::__detail, 159
- __hankel_uniform_olver
 - std::__detail, 160
- __hankel_uniform_outer
 - std::__detail, 160
- __hankel_uniform_sum
 - std::__detail, 160
- __heuman_lambda
 - std::__detail, 161
- __hurwitz_zeta
 - std::__detail, 161
- __hurwitz_zeta_euler_maclaurin
 - std::__detail, 162
- __hydrogen
 - std::__detail, 162
- __hyperg
 - std::__detail, 162
- __hyperg_luke
 - std::__detail, 163
- __hyperg_reflect
 - std::__detail, 163
- __hyperg_series
 - std::__detail, 163
- __jacobi_sncndn
 - std::__detail, 164
- __jacobi_zeta
 - std::__detail, 164
- __laguerre
 - std::__detail, 164
- __log_bincoef
 - std::__detail, 164
- __log_double_factorial
 - std::__detail, 165
- __log_factorial
 - std::__detail, 165
 - std::__detail::Factorial_table, 203
- __log_gamma
 - std::__detail, 165
- __log_gamma_bernoulli
 - std::__detail, 166
- __log_gamma_lanczos
 - std::__detail, 166
- __log_gamma_sign
 - std::__detail, 166
- __log_gamma_spouge
 - std::__detail, 167
- __log_pochhammer_l
 - std::__detail, 167
- __log_pochhammer_u
 - std::__detail, 168
- __logint
 - std::__detail, 168
- __n
 - std::__detail::Factorial_table, 203
- __owens_t
 - std::__detail, 168
- __pochhammer_l
 - std::__detail, 169
- __pochhammer_u

- std::__detail, 169
- __poly_hermite
 - std::__detail, 169
- __poly_hermite_asyp
 - std::__detail, 170
- __poly_hermite_recursion
 - std::__detail, 170
- __poly_jacobi
 - std::__detail, 170
- __poly_laguerre
 - std::__detail, 171
- __poly_laguerre_hyperg
 - std::__detail, 171
- __poly_laguerre_large_n
 - std::__detail, 173
- __poly_laguerre_recursion
 - std::__detail, 173
- __poly_legendre_p
 - std::__detail, 174
- __poly_legendre_q
 - std::__detail, 174
- __poly_radial_jacobi
 - std::__detail, 174
- __polylog
 - std::__detail, 174, 176
- __polylog_exp
 - std::__detail, 176
- __polylog_exp_asyp
 - std::__detail, 176
- __polylog_exp_int_neg
 - std::__detail, 177
- __polylog_exp_int_pos
 - std::__detail, 178
- __polylog_exp_neg
 - std::__detail, 178, 179
- __polylog_exp_neg_even
 - std::__detail, 179
- __polylog_exp_neg_odd
 - std::__detail, 180
- __polylog_exp_negative_real_part
 - std::__detail, 181
- __polylog_exp_pos
 - std::__detail, 181, 182
- __polylog_exp_real_neg
 - std::__detail, 183
- __polylog_exp_real_pos
 - std::__detail, 183, 184
- __psi
 - std::__detail, 184
- __psi_asyp
 - std::__detail, 185
- __psi_series
 - std::__detail, 185
- __riemann_zeta
 - std::__detail, 185
- __riemann_zeta_alt
 - std::__detail, 186
- __riemann_zeta_euler_maclaurin
 - std::__detail, 186
- __riemann_zeta_glob
 - std::__detail, 186
- __riemann_zeta_m_1
 - std::__detail, 186
- __riemann_zeta_m_1_sum
 - std::__detail, 188
- __riemann_zeta_product
 - std::__detail, 188
- __riemann_zeta_sum
 - std::__detail, 188
- __sinc
 - std::__detail, 189
- __sinc_pi
 - std::__detail, 189
- __sincosint
 - std::__detail, 189
- __sincosint_asyp
 - std::__detail, 189
- __sincosint_cont_frac
 - std::__detail, 190
- __sincosint_series
 - std::__detail, 190
- __sinhc
 - std::__detail, 190
- __sinhc_pi
 - std::__detail, 190
- __sinhint
 - std::__detail, 190
- __sph_bessel
 - std::__detail, 191
- __sph_bessel_ik
 - std::__detail, 191
- __sph_bessel_jn
 - std::__detail, 193
- __sph_hankel
 - std::__detail, 193
- __sph_hankel_1
 - std::__detail, 193, 194
- __sph_hankel_2
 - std::__detail, 194, 195
- __sph_harmonic
 - std::__detail, 195
- __sph_legendre
 - std::__detail, 195
- __sph_neumann
 - std::__detail, 196
- __students_t_cdf
 - std::__detail, 196
- __students_t_cdfc
 - std::__detail, 196

- std::__detail, 197
- __theta_1
 - std::__detail, 197
- __theta_2
 - std::__detail, 197
- __theta_2_asymp
 - std::__detail, 198
- __theta_2_sum
 - std::__detail, 198
- __theta_3
 - std::__detail, 198
- __theta_3_asymp
 - std::__detail, 198
- __theta_3_sum
 - std::__detail, 199
- __theta_4
 - std::__detail, 199
- __theta_c
 - std::__detail, 199
- __theta_d
 - std::__detail, 199
- __theta_n
 - std::__detail, 199
- __theta_s
 - std::__detail, 200
- __zernike
 - std::__detail, 200
- __znorm1
 - std::__detail, 200
- __znorm2
 - std::__detail, 200
- airy_ai
 - Extended Mathematical Special Functions, 19
- airy_aif
 - Extended Mathematical Special Functions, 19
- airy_ail
 - Extended Mathematical Special Functions, 19
- airy_bi
 - Extended Mathematical Special Functions, 19
- airy_bif
 - Extended Mathematical Special Functions, 19
- airy_bil
 - Extended Mathematical Special Functions, 20
- assoc_laguerre
 - Mathematical Special Functions, 69
- assoc_laguerref
 - Mathematical Special Functions, 69
- assoc_laguerrel
 - Mathematical Special Functions, 69
- assoc_legendre
 - Mathematical Special Functions, 69
- assoc_legendref
 - Mathematical Special Functions, 70
- assoc_legendrel
 - Mathematical Special Functions, 70
- bernoulli
 - Extended Mathematical Special Functions, 20
- bernoullif
 - Extended Mathematical Special Functions, 20
- bernoullil
 - Extended Mathematical Special Functions, 20
- beta
 - Mathematical Special Functions, 70
- betaf
 - Mathematical Special Functions, 70
- betal
 - Mathematical Special Functions, 71
- bincoef
 - Extended Mathematical Special Functions, 20
- bincoeff
 - Extended Mathematical Special Functions, 20
- bincoefl
 - Extended Mathematical Special Functions, 20
- bits/sf_airy.tcc, 205
- bits/sf_bessel.tcc, 207
- bits/sf_beta.tcc, 209
- bits/sf_cardinal.tcc, 211
- bits/sf_chebyshev.tcc, 213
- bits/sf_dawson.tcc, 215
- bits/sf_ellint.tcc, 216
- bits/sf_expint.tcc, 219
- bits/sf_fresnel.tcc, 221
- bits/sf_gamma.tcc, 222
- bits/sf_gegenbauer.tcc, 228
- bits/sf_hankel.tcc, 229
- bits/sf_hankel_new.tcc, 232
- bits/sf_hermite.tcc, 233
- bits/sf_hydrogen.tcc, 234
- bits/sf_hyperg.tcc, 235
- bits/sf_hypint.tcc, 237
- bits/sf_jacobi.tcc, 239
- bits/sf_laguerre.tcc, 240
- bits/sf_legendre.tcc, 242
- bits/sf_mod_bessel.tcc, 243
- bits/sf_owens_t.tcc, 246
- bits/sf_polylog.tcc, 247
- bits/sf_theta.tcc, 249
- bits/sf_trigint.tcc, 252
- bits/sf_zeta.tcc, 253
- bits/specfun.h, 256
- COSINT
 - std::__detail, 112
- chebyshev_t
 - Extended Mathematical Special Functions, 20
- chebyshev_tf
 - Extended Mathematical Special Functions, 21

- chebyshev_tl
 - Extended Mathematical Special Functions, [21](#)
- chebyshev_u
 - Extended Mathematical Special Functions, [21](#)
- chebyshev_uf
 - Extended Mathematical Special Functions, [21](#)
- chebyshev_ul
 - Extended Mathematical Special Functions, [21](#)
- chebyshev_v
 - Extended Mathematical Special Functions, [22](#)
- chebyshev_vf
 - Extended Mathematical Special Functions, [22](#)
- chebyshev_vl
 - Extended Mathematical Special Functions, [22](#)
- chebyshev_w
 - Extended Mathematical Special Functions, [22](#)
- chebyshev_wf
 - Extended Mathematical Special Functions, [23](#)
- chebyshev_wl
 - Extended Mathematical Special Functions, [23](#)
- clausen
 - Extended Mathematical Special Functions, [23](#)
- clausen_c
 - Extended Mathematical Special Functions, [23](#)
- clausen_cf
 - Extended Mathematical Special Functions, [24](#)
- clausen_cl
 - Extended Mathematical Special Functions, [24](#)
- clausen_s
 - Extended Mathematical Special Functions, [24](#)
- clausen_sf
 - Extended Mathematical Special Functions, [24](#)
- clausen_sl
 - Extended Mathematical Special Functions, [24](#)
- clausenf
 - Extended Mathematical Special Functions, [25](#)
- clausenl
 - Extended Mathematical Special Functions, [25](#)
- comp_ellint_1
 - Mathematical Special Functions, [71](#)
- comp_ellint_1f
 - Mathematical Special Functions, [71](#)
- comp_ellint_1l
 - Mathematical Special Functions, [71](#)
- comp_ellint_2
 - Mathematical Special Functions, [71](#)
- comp_ellint_2f
 - Mathematical Special Functions, [72](#)
- comp_ellint_2l
 - Mathematical Special Functions, [72](#)
- comp_ellint_3
 - Mathematical Special Functions, [72](#)
- comp_ellint_3f
 - Mathematical Special Functions, [73](#)
- comp_ellint_3l
 - Mathematical Special Functions, [73](#)
- comp_ellint_d
 - Extended Mathematical Special Functions, [25](#)
- comp_ellint_df
 - Extended Mathematical Special Functions, [26](#)
- comp_ellint_dl
 - Extended Mathematical Special Functions, [26](#)
- comp_ellint_rf
 - Extended Mathematical Special Functions, [26](#)
- comp_ellint_rg
 - Extended Mathematical Special Functions, [26](#)
- conf_hyperg
 - Extended Mathematical Special Functions, [27](#)
- conf_hyperg_lim
 - Extended Mathematical Special Functions, [27](#)
- conf_hyperg_limf
 - Extended Mathematical Special Functions, [27](#)
- conf_hyperg_liml
 - Extended Mathematical Special Functions, [27](#)
- conf_hypergf
 - Extended Mathematical Special Functions, [28](#)
- conf_hypergl
 - Extended Mathematical Special Functions, [28](#)
- coshint
 - Extended Mathematical Special Functions, [28](#)
- coshintf
 - Extended Mathematical Special Functions, [28](#)
- coshintl
 - Extended Mathematical Special Functions, [28](#)
- cosint
 - Extended Mathematical Special Functions, [29](#)
- cosintf
 - Extended Mathematical Special Functions, [29](#)
- cosintl
 - Extended Mathematical Special Functions, [29](#)
- cyl_bessel_i
 - Mathematical Special Functions, [73](#)
- cyl_bessel_if
 - Mathematical Special Functions, [73](#)
- cyl_bessel_il
 - Mathematical Special Functions, [73](#)
- cyl_bessel_j
 - Mathematical Special Functions, [74](#)
- cyl_bessel_jf
 - Mathematical Special Functions, [74](#)
- cyl_bessel_jl
 - Mathematical Special Functions, [74](#)
- cyl_bessel_k
 - Mathematical Special Functions, [74](#)
- cyl_bessel_kf
 - Mathematical Special Functions, [75](#)
- cyl_bessel_kl
 - Mathematical Special Functions, [75](#)

- cyl_hankel_1
 - Extended Mathematical Special Functions, [29](#)
- cyl_hankel_1f
 - Extended Mathematical Special Functions, [31](#)
- cyl_hankel_1l
 - Extended Mathematical Special Functions, [31](#)
- cyl_hankel_2
 - Extended Mathematical Special Functions, [31](#)
- cyl_hankel_2f
 - Extended Mathematical Special Functions, [32](#)
- cyl_hankel_2l
 - Extended Mathematical Special Functions, [32](#)
- cyl_neumann
 - Mathematical Special Functions, [75](#)
- cyl_neumannf
 - Mathematical Special Functions, [75](#)
- cyl_neumannl
 - Mathematical Special Functions, [76](#)
- dawson
 - Extended Mathematical Special Functions, [32](#)
- dawsonf
 - Extended Mathematical Special Functions, [33](#)
- dawsonl
 - Extended Mathematical Special Functions, [33](#)
- digamma
 - Extended Mathematical Special Functions, [33](#)
- digammaf
 - Extended Mathematical Special Functions, [33](#)
- digammal
 - Extended Mathematical Special Functions, [33](#)
- dilog
 - Extended Mathematical Special Functions, [33](#)
- dilogf
 - Extended Mathematical Special Functions, [33](#)
- dilogl
 - Extended Mathematical Special Functions, [33](#)
- dirichlet_beta
 - Extended Mathematical Special Functions, [33](#)
- dirichlet_betaf
 - Extended Mathematical Special Functions, [34](#)
- dirichlet_betal
 - Extended Mathematical Special Functions, [34](#)
- dirichlet_eta
 - Extended Mathematical Special Functions, [34](#)
- dirichlet_etaf
 - Extended Mathematical Special Functions, [34](#)
- dirichlet_etalf
 - Extended Mathematical Special Functions, [34](#)
- dirichlet_etall
 - Extended Mathematical Special Functions, [34](#)
- double_factorial
 - Extended Mathematical Special Functions, [34](#)
- double_factorialf
 - Extended Mathematical Special Functions, [35](#)
- double_factoriall
 - Extended Mathematical Special Functions, [35](#)
- ellint_1
 - Mathematical Special Functions, [76](#)
- ellint_1f
 - Mathematical Special Functions, [76](#)
- ellint_1l
 - Mathematical Special Functions, [76](#)
- ellint_2
 - Mathematical Special Functions, [77](#)
- ellint_2f
 - Mathematical Special Functions, [77](#)
- ellint_2l
 - Mathematical Special Functions, [77](#)
- ellint_3
 - Mathematical Special Functions, [77](#)
- ellint_3f
 - Mathematical Special Functions, [78](#)
- ellint_3l
 - Mathematical Special Functions, [78](#)
- ellint_cel
 - Extended Mathematical Special Functions, [35](#)
- ellint_celf
 - Extended Mathematical Special Functions, [35](#)
- ellint_cell
 - Extended Mathematical Special Functions, [35](#)
- ellint_d
 - Extended Mathematical Special Functions, [35](#)
- ellint_df
 - Extended Mathematical Special Functions, [36](#)
- ellint_dl
 - Extended Mathematical Special Functions, [36](#)
- ellint_el1
 - Extended Mathematical Special Functions, [36](#)
- ellint_el1f
 - Extended Mathematical Special Functions, [36](#)
- ellint_el1l
 - Extended Mathematical Special Functions, [36](#)
- ellint_el2
 - Extended Mathematical Special Functions, [36](#)
- ellint_el2f
 - Extended Mathematical Special Functions, [37](#)
- ellint_el2l
 - Extended Mathematical Special Functions, [37](#)
- ellint_el3
 - Extended Mathematical Special Functions, [37](#)
- ellint_el3f
 - Extended Mathematical Special Functions, [37](#)
- ellint_el3l
 - Extended Mathematical Special Functions, [37](#)
- ellint_rc
 - Extended Mathematical Special Functions, [37](#)
- ellint_rcf
 - Extended Mathematical Special Functions, [38](#)

- ellint_rcl
 - Extended Mathematical Special Functions, [38](#)
- ellint_rd
 - Extended Mathematical Special Functions, [38](#)
- ellint_rdf
 - Extended Mathematical Special Functions, [39](#)
- ellint_rdl
 - Extended Mathematical Special Functions, [39](#)
- ellint_rf
 - Extended Mathematical Special Functions, [39](#)
- ellint_rff
 - Extended Mathematical Special Functions, [39](#)
- ellint_rfl
 - Extended Mathematical Special Functions, [39](#)
- ellint_rg
 - Extended Mathematical Special Functions, [40](#)
- ellint_rgf
 - Extended Mathematical Special Functions, [40](#)
- ellint_rgl
 - Extended Mathematical Special Functions, [40](#)
- ellint_rj
 - Extended Mathematical Special Functions, [40](#)
- ellint_rjf
 - Extended Mathematical Special Functions, [41](#)
- ellint_rjl
 - Extended Mathematical Special Functions, [41](#)
- ellnome
 - Extended Mathematical Special Functions, [41](#)
- ellnomef
 - Extended Mathematical Special Functions, [42](#)
- ellnomel
 - Extended Mathematical Special Functions, [42](#)
- evenzeta
 - std::__detail, [200](#)
- expint
 - Mathematical Special Functions, [78](#)
- expint_e1
 - Extended Mathematical Special Functions, [42](#)
- expint_e1f
 - Extended Mathematical Special Functions, [42](#)
- expint_e1l
 - Extended Mathematical Special Functions, [42](#)
- expint_en
 - Extended Mathematical Special Functions, [42](#)
- expint_enf
 - Extended Mathematical Special Functions, [42](#)
- expint_enl
 - Extended Mathematical Special Functions, [42](#)
- expintf
 - Mathematical Special Functions, [79](#)
- expintl
 - Mathematical Special Functions, [79](#)
- Extended Mathematical Special Functions, [11](#)
 - _GLIBCXX_JACOBI_CN, [19](#)
 - _GLIBCXX_JACOBI_DN, [19](#)
 - _GLIBCXX_JACOBI_SN, [19](#)
 - airy_ai, [19](#)
 - airy_aif, [19](#)
 - airy_ail, [19](#)
 - airy_bi, [19](#)
 - airy_bif, [19](#)
 - airy_bil, [20](#)
 - bernoulli, [20](#)
 - bernoullif, [20](#)
 - bernoullil, [20](#)
 - bincoef, [20](#)
 - bincoeff, [20](#)
 - bincoefl, [20](#)
 - chebyshev_t, [20](#)
 - chebyshev_tf, [21](#)
 - chebyshev_tl, [21](#)
 - chebyshev_u, [21](#)
 - chebyshev_uf, [21](#)
 - chebyshev_ul, [21](#)
 - chebyshev_v, [22](#)
 - chebyshev_vf, [22](#)
 - chebyshev_vl, [22](#)
 - chebyshev_w, [22](#)
 - chebyshev_wf, [23](#)
 - chebyshev_wl, [23](#)
 - clausen, [23](#)
 - clausen_c, [23](#)
 - clausen_cf, [24](#)
 - clausen_cl, [24](#)
 - clausen_s, [24](#)
 - clausen_sf, [24](#)
 - clausen_sl, [24](#)
 - clausenf, [25](#)
 - clausenl, [25](#)
 - comp_ellint_d, [25](#)
 - comp_ellint_df, [26](#)
 - comp_ellint_dl, [26](#)
 - comp_ellint_rf, [26](#)
 - comp_ellint_rg, [26](#)
 - conf_hyperg, [27](#)
 - conf_hyperg_lim, [27](#)
 - conf_hyperg_limf, [27](#)
 - conf_hyperg_liml, [27](#)
 - conf_hypergf, [28](#)
 - conf_hypergl, [28](#)
 - coshint, [28](#)
 - coshintf, [28](#)
 - coshintl, [28](#)
 - cosint, [29](#)
 - cosintf, [29](#)
 - cosintl, [29](#)
 - cyl_hankel_1, [29](#)
 - cyl_hankel_1f, [31](#)

cyl_hankel_1l, 31
 cyl_hankel_2, 31
 cyl_hankel_2f, 32
 cyl_hankel_2l, 32
 dawson, 32
 dawsonf, 33
 dawsonl, 33
 digamma, 33
 digammaf, 33
 digammal, 33
 dilog, 33
 dilogf, 33
 dilogl, 33
 dirichlet_beta, 33
 dirichlet_betaf, 34
 dirichlet_betal, 34
 dirichlet_eta, 34
 dirichlet_etaf, 34
 dirichlet_etall, 34
 double_factorial, 34
 double_factorialf, 35
 double_factoriall, 35
 ellint_cel, 35
 ellint_celf, 35
 ellint_cell, 35
 ellint_d, 35
 ellint_df, 36
 ellint_dl, 36
 ellint_el1, 36
 ellint_el1f, 36
 ellint_el1l, 36
 ellint_el2, 36
 ellint_el2f, 37
 ellint_el2l, 37
 ellint_el3, 37
 ellint_el3f, 37
 ellint_el3l, 37
 ellint_rc, 37
 ellint_rcf, 38
 ellint_rcl, 38
 ellint_rd, 38
 ellint_rdf, 39
 ellint_rdl, 39
 ellint_rf, 39
 ellint_rff, 39
 ellint_rfl, 39
 ellint_rg, 40
 ellint_rgf, 40
 ellint_rgl, 40
 ellint_rj, 40
 ellint_rjf, 41
 ellint_rjl, 41
 ellnome, 41
 ellnomef, 42
 ellnomel, 42
 expint_e1, 42
 expint_e1f, 42
 expint_e1l, 42
 expint_en, 42
 expint_enf, 42
 expint_enl, 42
 factorial, 42
 factorialf, 42
 factoriall, 42
 fresnel_c, 43
 fresnel_cf, 43
 fresnel_cl, 43
 fresnel_s, 43
 fresnel_sf, 43
 fresnel_sl, 43
 gamma_l, 43
 gamma_lf, 43
 gamma_ll, 44
 gamma_p, 44
 gamma_pf, 44
 gamma_pl, 44
 gamma_q, 44
 gamma_qf, 44
 gamma_ql, 44
 gamma_u, 44
 gamma_uf, 44
 gamma_ul, 44
 gegenbauer, 44
 gegenbauerf, 45
 gegenbauerl, 45
 heuman_lambda, 45
 heuman_lambdaf, 45
 heuman_lambdal, 45
 hurwitz_zeta, 45
 hurwitz_zetaf, 46
 hurwitz_zetal, 46
 hyperg, 46
 hypergf, 46
 hypergl, 46
 ibeta, 46
 ibetac, 47
 ibetacf, 47
 ibetacl, 47
 ibetaf, 47
 ibetal, 48
 jacobi, 48
 jacobi_cn, 48
 jacobi_cnf, 48
 jacobi_cnl, 48
 jacobi_dn, 48
 jacobi_dnf, 49
 jacobi_dnl, 49
 jacobi_sn, 49

jacobi_snf, [49](#)
 jacobi_snl, [50](#)
 jacobi_zeta, [50](#)
 jacobi_zetaf, [50](#)
 jacobi_zetal, [50](#)
 jacobif, [50](#)
 jacobil, [50](#)
 lbincoef, [50](#)
 lbincoeff, [51](#)
 lbincoefl, [51](#)
 ldouble_factorial, [51](#)
 ldouble_factorialf, [51](#)
 ldouble_factoriall, [51](#)
 legendre_q, [51](#)
 legendre_qf, [51](#)
 legendre_ql, [51](#)
 lfactorial, [51](#)
 lfactorialf, [51](#)
 lfactoriall, [51](#)
 logint, [52](#)
 logintf, [52](#)
 logintl, [52](#)
 lpochhammer_l, [52](#)
 lpochhammer_lf, [52](#)
 lpochhammer_ll, [52](#)
 lpochhammer_u, [52](#)
 lpochhammer_uf, [53](#)
 lpochhammer_ul, [53](#)
 owens_t, [53](#)
 owens_tf, [53](#)
 owens_tl, [53](#)
 pochhammer_l, [53](#)
 pochhammer_lf, [54](#)
 pochhammer_ll, [54](#)
 pochhammer_u, [54](#)
 pochhammer_uf, [54](#)
 pochhammer_ul, [54](#)
 polylog, [54](#)
 polylogf, [54](#)
 polylogl, [54](#)
 psi, [55](#)
 psif, [55](#)
 psil, [55](#)
 radpoly, [55](#)
 radpolyf, [55](#)
 radpolyl, [55](#)
 sinc, [55](#)
 sinc_pi, [55](#)
 sinc_pif, [56](#)
 sinc_pil, [56](#)
 sincf, [56](#)
 sincl, [56](#)
 sinhc, [56](#)
 sinhc_pi, [56](#)
 sinhc_pif, [56](#)
 sinhc_pil, [56](#)
 sinhcf, [56](#)
 sinhcl, [56](#)
 sinhint, [56](#)
 sinhintf, [57](#)
 sinhintl, [57](#)
 sinint, [57](#)
 sinintf, [57](#)
 sinintl, [57](#)
 sph_bessel_i, [58](#)
 sph_bessel_if, [58](#)
 sph_bessel_il, [58](#)
 sph_bessel_k, [58](#)
 sph_bessel_kf, [58](#)
 sph_bessel_kl, [58](#)
 sph_hankel_1, [58](#)
 sph_hankel_1f, [59](#)
 sph_hankel_1l, [59](#)
 sph_hankel_2, [59](#)
 sph_hankel_2f, [59](#)
 sph_hankel_2l, [60](#)
 sph_harmonic, [60](#)
 sph_harmonicf, [60](#)
 sph_harmonicl, [60](#)
 theta_1, [61](#)
 theta_1f, [61](#)
 theta_1l, [61](#)
 theta_2, [61](#)
 theta_2f, [62](#)
 theta_2l, [62](#)
 theta_3, [62](#)
 theta_3f, [62](#)
 theta_3l, [62](#)
 theta_4, [63](#)
 theta_4f, [63](#)
 theta_4l, [63](#)
 theta_c, [63](#)
 theta_cf, [64](#)
 theta_cl, [64](#)
 theta_d, [64](#)
 theta_df, [64](#)
 theta_dl, [64](#)
 theta_n, [65](#)
 theta_nf, [65](#)
 theta_nl, [65](#)
 theta_s, [65](#)
 theta_sf, [66](#)
 theta_sl, [66](#)
 zernike, [66](#)
 zernikef, [66](#)
 zernikel, [66](#)
 factorial

- Extended Mathematical Special Functions, [42](#)
- factorialf
 - Extended Mathematical Special Functions, [42](#)
- factoriall
 - Extended Mathematical Special Functions, [42](#)
- fresnel_c
 - Extended Mathematical Special Functions, [43](#)
- fresnel_cf
 - Extended Mathematical Special Functions, [43](#)
- fresnel_cl
 - Extended Mathematical Special Functions, [43](#)
- fresnel_s
 - Extended Mathematical Special Functions, [43](#)
- fresnel_sf
 - Extended Mathematical Special Functions, [43](#)
- fresnel_sl
 - Extended Mathematical Special Functions, [43](#)
- gamma_l
 - Extended Mathematical Special Functions, [43](#)
- gamma_lf
 - Extended Mathematical Special Functions, [43](#)
- gamma_ll
 - Extended Mathematical Special Functions, [44](#)
- gamma_p
 - Extended Mathematical Special Functions, [44](#)
- gamma_pf
 - Extended Mathematical Special Functions, [44](#)
- gamma_pl
 - Extended Mathematical Special Functions, [44](#)
- gamma_q
 - Extended Mathematical Special Functions, [44](#)
- gamma_qf
 - Extended Mathematical Special Functions, [44](#)
- gamma_ql
 - Extended Mathematical Special Functions, [44](#)
- gamma_u
 - Extended Mathematical Special Functions, [44](#)
- gamma_uf
 - Extended Mathematical Special Functions, [44](#)
- gamma_ul
 - Extended Mathematical Special Functions, [44](#)
- gegenbauer
 - Extended Mathematical Special Functions, [44](#)
- gegenbauerf
 - Extended Mathematical Special Functions, [45](#)
- gegenbauerl
 - Extended Mathematical Special Functions, [45](#)
- hermite
 - Mathematical Special Functions, [79](#)
- hermitef
 - Mathematical Special Functions, [79](#)
- hermitel
 - Mathematical Special Functions, [79](#)
- heuman_lambda
 - Extended Mathematical Special Functions, [45](#)
- heuman_lambdaf
 - Extended Mathematical Special Functions, [45](#)
- heuman_lambdal
 - Extended Mathematical Special Functions, [45](#)
- hurwitz_zeta
 - Extended Mathematical Special Functions, [45](#)
- hurwitz_zetaf
 - Extended Mathematical Special Functions, [46](#)
- hurwitz_zetal
 - Extended Mathematical Special Functions, [46](#)
- hyperg
 - Extended Mathematical Special Functions, [46](#)
- hypergf
 - Extended Mathematical Special Functions, [46](#)
- hypergl
 - Extended Mathematical Special Functions, [46](#)
- ibeta
 - Extended Mathematical Special Functions, [46](#)
- ibetac
 - Extended Mathematical Special Functions, [47](#)
- ibetacf
 - Extended Mathematical Special Functions, [47](#)
- ibetacl
 - Extended Mathematical Special Functions, [47](#)
- ibetaf
 - Extended Mathematical Special Functions, [47](#)
- ibetal
 - Extended Mathematical Special Functions, [48](#)
- jacobi
 - Extended Mathematical Special Functions, [48](#)
- jacobi_cn
 - Extended Mathematical Special Functions, [48](#)
- jacobi_cnf
 - Extended Mathematical Special Functions, [48](#)
- jacobi_cnl
 - Extended Mathematical Special Functions, [48](#)
- jacobi_dn
 - Extended Mathematical Special Functions, [48](#)
- jacobi_dnf
 - Extended Mathematical Special Functions, [49](#)
- jacobi_dnl
 - Extended Mathematical Special Functions, [49](#)
- jacobi_sn
 - Extended Mathematical Special Functions, [49](#)
- jacobi_snf
 - Extended Mathematical Special Functions, [49](#)
- jacobi_snl
 - Extended Mathematical Special Functions, [50](#)
- jacobi_zeta
 - Extended Mathematical Special Functions, [50](#)
- jacobi_zetaf
 - Extended Mathematical Special Functions, [50](#)

- Extended Mathematical Special Functions, [50](#)
- `jacobi_zetal`
 - Extended Mathematical Special Functions, [50](#)
- `jacobiif`
 - Extended Mathematical Special Functions, [50](#)
- `jacobil`
 - Extended Mathematical Special Functions, [50](#)
- `laguerre`
 - Mathematical Special Functions, [80](#)
- `laguerref`
 - Mathematical Special Functions, [80](#)
- `laguerrel`
 - Mathematical Special Functions, [80](#)
- `lbincoef`
 - Extended Mathematical Special Functions, [50](#)
- `lbincoeff`
 - Extended Mathematical Special Functions, [51](#)
- `lbincoefl`
 - Extended Mathematical Special Functions, [51](#)
- `ldouble_factorial`
 - Extended Mathematical Special Functions, [51](#)
- `ldouble_factorialf`
 - Extended Mathematical Special Functions, [51](#)
- `ldouble_factoriall`
 - Extended Mathematical Special Functions, [51](#)
- `legendre`
 - Mathematical Special Functions, [80](#)
- `legendre_q`
 - Extended Mathematical Special Functions, [51](#)
- `legendre_qf`
 - Extended Mathematical Special Functions, [51](#)
- `legendre_ql`
 - Extended Mathematical Special Functions, [51](#)
- `legendref`
 - Mathematical Special Functions, [81](#)
- `legendrel`
 - Mathematical Special Functions, [81](#)
- `lfactorial`
 - Extended Mathematical Special Functions, [51](#)
- `lfactorialf`
 - Extended Mathematical Special Functions, [51](#)
- `lfactoriall`
 - Extended Mathematical Special Functions, [51](#)
- `logint`
 - Extended Mathematical Special Functions, [52](#)
- `logintf`
 - Extended Mathematical Special Functions, [52](#)
- `logintl`
 - Extended Mathematical Special Functions, [52](#)
- `lpochhammer_l`
 - Extended Mathematical Special Functions, [52](#)
- `lpochhammer_lf`
 - Extended Mathematical Special Functions, [52](#)
- `lpochhammer_ll`
 - Extended Mathematical Special Functions, [52](#)
- `lpochhammer_u`
 - Extended Mathematical Special Functions, [52](#)
- `lpochhammer_uf`
 - Extended Mathematical Special Functions, [53](#)
- `lpochhammer_ul`
 - Extended Mathematical Special Functions, [53](#)
- Mathematical Special Functions, [67](#)
 - `assoc_laguerre`, [69](#)
 - `assoc_laguerref`, [69](#)
 - `assoc_laguerrel`, [69](#)
 - `assoc_legendre`, [69](#)
 - `assoc_legendref`, [70](#)
 - `assoc_legendrel`, [70](#)
 - `beta`, [70](#)
 - `betaf`, [70](#)
 - `betal`, [71](#)
 - `comp_ellint_1`, [71](#)
 - `comp_ellint_1f`, [71](#)
 - `comp_ellint_1l`, [71](#)
 - `comp_ellint_2`, [71](#)
 - `comp_ellint_2f`, [72](#)
 - `comp_ellint_2l`, [72](#)
 - `comp_ellint_3`, [72](#)
 - `comp_ellint_3f`, [73](#)
 - `comp_ellint_3l`, [73](#)
 - `cyl_bessel_i`, [73](#)
 - `cyl_bessel_if`, [73](#)
 - `cyl_bessel_il`, [73](#)
 - `cyl_bessel_j`, [74](#)
 - `cyl_bessel_jf`, [74](#)
 - `cyl_bessel_jl`, [74](#)
 - `cyl_bessel_k`, [74](#)
 - `cyl_bessel_kf`, [75](#)
 - `cyl_bessel_kl`, [75](#)
 - `cyl_neumann`, [75](#)
 - `cyl_neumannf`, [75](#)
 - `cyl_neumannl`, [76](#)
 - `ellint_1`, [76](#)
 - `ellint_1f`, [76](#)
 - `ellint_1l`, [76](#)
 - `ellint_2`, [77](#)
 - `ellint_2f`, [77](#)
 - `ellint_2l`, [77](#)
 - `ellint_3`, [77](#)
 - `ellint_3f`, [78](#)
 - `ellint_3l`, [78](#)
 - `expint`, [78](#)
 - `expintf`, [79](#)
 - `expintl`, [79](#)
 - `hermite`, [79](#)
 - `hermitef`, [79](#)

- hermitel, [79](#)
- laguerre, [80](#)
- laguerref, [80](#)
- laguerrel, [80](#)
- legendre, [80](#)
- legendref, [81](#)
- legendrel, [81](#)
- riemann_zeta, [81](#)
- riemann_zetaf, [81](#)
- riemann_zetal, [81](#)
- sph_bessel, [82](#)
- sph_besself, [82](#)
- sph_bessell, [82](#)
- sph_legendre, [82](#)
- sph_legendref, [83](#)
- sph_legendrel, [83](#)
- sph_neumann, [83](#)
- sph_neumannf, [83](#)
- sph_neumannl, [83](#)

- owens_t
 - Extended Mathematical Special Functions, [53](#)
- owens_tf
 - Extended Mathematical Special Functions, [53](#)
- owens_tl
 - Extended Mathematical Special Functions, [53](#)

- pochhammer_l
 - Extended Mathematical Special Functions, [53](#)
- pochhammer_lf
 - Extended Mathematical Special Functions, [54](#)
- pochhammer_ll
 - Extended Mathematical Special Functions, [54](#)
- pochhammer_u
 - Extended Mathematical Special Functions, [54](#)
- pochhammer_uf
 - Extended Mathematical Special Functions, [54](#)
- pochhammer_ul
 - Extended Mathematical Special Functions, [54](#)
- polylog
 - Extended Mathematical Special Functions, [54](#)
- polylogf
 - Extended Mathematical Special Functions, [54](#)
- polylogl
 - Extended Mathematical Special Functions, [54](#)
- psi
 - Extended Mathematical Special Functions, [55](#)
- psif
 - Extended Mathematical Special Functions, [55](#)
- psil
 - Extended Mathematical Special Functions, [55](#)

- radpoly
 - Extended Mathematical Special Functions, [55](#)
- radpolyf
 - Extended Mathematical Special Functions, [55](#)
- riemann_zeta
 - Mathematical Special Functions, [81](#)
- riemann_zetaf
 - Mathematical Special Functions, [81](#)
- riemann_zetal
 - Mathematical Special Functions, [81](#)

- SININT
 - std::__detail, [112](#)
- sf_airy.tcc
 - _GLIBCXX_BITS_SF_AIRY_TCC, [207](#)
- sf_bessel.tcc
 - _GLIBCXX_BITS_SF_BESSEL_TCC, [209](#)
- sf_beta.tcc
 - _GLIBCXX_BITS_SF_BETA_TCC, [211](#)
- sf_cardinal.tcc
 - _GLIBCXX_BITS_SF_CARDINAL_TCC, [213](#)
- sf_chebyshev.tcc
 - _GLIBCXX_SF_CHEBYSHEV_TCC, [214](#)
- sf_dawson.tcc
 - _GLIBCXX_SF_DAWSON_TCC, [216](#)
- sf_ellint.tcc
 - _GLIBCXX_BITS_SF_ELLINT_TCC, [218](#)
- sf_expint.tcc
 - _GLIBCXX_BITS_SF_EXPINT_TCC, [221](#)
- sf_fresnel.tcc
 - _GLIBCXX_SF_FRESNEL_TCC, [222](#)
- sf_gamma.tcc
 - _GLIBCXX_BITS_SF_GAMMA_TCC, [228](#)
- sf_gegenbauer.tcc
 - _GLIBCXX_SF_GEGENBAUER_TCC, [229](#)
- sf_hankel.tcc
 - _GLIBCXX_BITS_SF_HANKEL_TCC, [232](#)
- sf_hankel_new.tcc
 - _GLIBCXX_BITS_SF_HANKEL_NEW_TCC, [232](#)
- sf_hermite.tcc
 - _GLIBCXX_BITS_SF_HERMITE_TCC, [234](#)
- sf_hydrogen.tcc
 - _GLIBCXX_BITS_SF_HYDROGEN_TCC, [235](#)
- sf_hyperg.tcc
 - _GLIBCXX_BITS_SF_HYPERG_TCC, [237](#)
- sf_hypint.tcc
 - _GLIBCXX_SF_HYPINT_TCC, [238](#)
- sf_jacobi.tcc
 - _GLIBCXX_SF_JACOBI_TCC, [240](#)
- sf_laguerre.tcc
 - _GLIBCXX_BITS_SF_LAGUERRE_TCC, [242](#)
- sf_legendre.tcc
 - _GLIBCXX_BITS_SF_LEGENDRE_TCC, [243](#)
- sf_mod_bessel.tcc
 - _GLIBCXX_BITS_SF_MOD_BESSEL_TCC, [245](#)

- sf_owens.tcc
 - _GLIBCXX_BITS_SF_OWENS_T_TCC, [246](#)
- sf_polylog.tcc
 - _GLIBCXX_BITS_SF_POLYLOG_TCC, [249](#)
- sf_theta.tcc
 - _GLIBCXX_SF_THETA_TCC, [251](#)
- sf_trigint.tcc
 - _GLIBCXX_SF_TRIGINT_TCC, [253](#)
- sf_zeta.tcc
 - _GLIBCXX_BITS_SF_ZETA_TCC, [255](#)
- sinc
 - Extended Mathematical Special Functions, [55](#)
- sinc_pi
 - Extended Mathematical Special Functions, [55](#)
- sinc_pif
 - Extended Mathematical Special Functions, [56](#)
- sinc_pil
 - Extended Mathematical Special Functions, [56](#)
- sincf
 - Extended Mathematical Special Functions, [56](#)
- sincl
 - Extended Mathematical Special Functions, [56](#)
- sinhc
 - Extended Mathematical Special Functions, [56](#)
- sinhc_pi
 - Extended Mathematical Special Functions, [56](#)
- sinhc_pif
 - Extended Mathematical Special Functions, [56](#)
- sinhc_pil
 - Extended Mathematical Special Functions, [56](#)
- sinhcf
 - Extended Mathematical Special Functions, [56](#)
- sinhcl
 - Extended Mathematical Special Functions, [56](#)
- sinhint
 - Extended Mathematical Special Functions, [56](#)
- sinhintf
 - Extended Mathematical Special Functions, [57](#)
- sinhintl
 - Extended Mathematical Special Functions, [57](#)
- sinint
 - Extended Mathematical Special Functions, [57](#)
- sinintf
 - Extended Mathematical Special Functions, [57](#)
- sinintl
 - Extended Mathematical Special Functions, [57](#)
- specfun.h
 - _STDCPP_MATH_SPEC_FUNCS_, [266](#)
 - _cpp_lib_math_special_functions, [266](#)
- sph_bessel
 - Mathematical Special Functions, [82](#)
- sph_bessel_i
 - Extended Mathematical Special Functions, [58](#)
- sph_bessel_if
 - Extended Mathematical Special Functions, [58](#)
- sph_bessel_il
 - Extended Mathematical Special Functions, [58](#)
- sph_bessel_k
 - Extended Mathematical Special Functions, [58](#)
- sph_bessel_kf
 - Extended Mathematical Special Functions, [58](#)
- sph_bessel_kl
 - Extended Mathematical Special Functions, [58](#)
- sph_besself
 - Mathematical Special Functions, [82](#)
- sph_bessell
 - Mathematical Special Functions, [82](#)
- sph_hankel_1
 - Extended Mathematical Special Functions, [58](#)
- sph_hankel_1f
 - Extended Mathematical Special Functions, [59](#)
- sph_hankel_1l
 - Extended Mathematical Special Functions, [59](#)
- sph_hankel_2
 - Extended Mathematical Special Functions, [59](#)
- sph_hankel_2f
 - Extended Mathematical Special Functions, [59](#)
- sph_hankel_2l
 - Extended Mathematical Special Functions, [60](#)
- sph_harmonic
 - Extended Mathematical Special Functions, [60](#)
- sph_harmonicf
 - Extended Mathematical Special Functions, [60](#)
- sph_harmonicl
 - Extended Mathematical Special Functions, [60](#)
- sph_legendre
 - Mathematical Special Functions, [82](#)
- sph_legendref
 - Mathematical Special Functions, [83](#)
- sph_legendrel
 - Mathematical Special Functions, [83](#)
- sph_neumann
 - Mathematical Special Functions, [83](#)
- sph_neumannf
 - Mathematical Special Functions, [83](#)
- sph_neumannl
 - Mathematical Special Functions, [83](#)
- std, [93](#)
- std::__detail, [95](#)
 - _Num_Euler_Maclaurin_zeta, [200](#)
 - _S_Euler_Maclaurin_zeta, [201](#)
 - _S_double_factorial_table, [201](#)
 - _S_factorial_table, [201](#)
 - _S_neg_double_factorial_table, [201](#)
 - _S_num_double_factorials, [201](#)
 - _S_num_double_factorials< double >, [201](#)
 - _S_num_double_factorials< float >, [201](#)
 - _S_num_double_factorials< long double >, [201](#)

- [__S_num_factorials](#), 201
- [__S_num_factorials< double >](#), 201
- [__S_num_factorials< float >](#), 202
- [__S_num_factorials< long double >](#), 202
- [__S_num_neg_double_factorials](#), 202
- [__S_num_neg_double_factorials< double >](#), 202
- [__S_num_neg_double_factorials< float >](#), 202
- [__S_num_neg_double_factorials< long double >](#), 202
- [__S_num_zetam1](#), 202
- [__S_zetam1](#), 202
- [__airy](#), 112, 113
- [__airy_ai](#), 115
- [__airy_arg](#), 115
- [__airy_asymp_absarg_ge_pio3](#), 115
- [__airy_asymp_absarg_lt_pio3](#), 116
- [__airy_bessel_i](#), 116
- [__airy_bessel_k](#), 117
- [__airy_bi](#), 118
- [__airy_hyperg_rational](#), 118
- [__assoc_laguerre](#), 119
- [__assoc_legendre_p](#), 119
- [__bernoulli](#), 119
- [__bernoulli_2n](#), 120
- [__bernoulli_series](#), 120
- [__beta](#), 120
- [__beta_gamma](#), 121
- [__beta_inc](#), 121
- [__beta_inc_cont_frac](#), 122
- [__beta_lgamma](#), 122
- [__beta_product](#), 122
- [__bincoef](#), 123
- [__binomial_cdf](#), 123
- [__binomial_cdfc](#), 123
- [__bose_einstein](#), 124
- [__chebyshev_recur](#), 124
- [__chebyshev_t](#), 124
- [__chebyshev_u](#), 124
- [__chebyshev_v](#), 124
- [__chebyshev_w](#), 124
- [__chi_squared_pdf](#), 125
- [__chi_squared_pdfc](#), 125
- [__chshint](#), 125
- [__chshint_cont_frac](#), 125
- [__chshint_series](#), 126
- [__clamp_0_m2pi](#), 126
- [__clamp_pi](#), 126
- [__clausen](#), 126
- [__clausen_c](#), 127
- [__clausen_s](#), 127, 128
- [__comp_ellint_1](#), 128
- [__comp_ellint_2](#), 129
- [__comp_ellint_3](#), 129
- [__comp_ellint_d](#), 129
- [__comp_ellint_rf](#), 130
- [__comp_ellint_rg](#), 130
- [__conf_hyperg](#), 130
- [__conf_hyperg_lim](#), 130
- [__conf_hyperg_lim_series](#), 130
- [__conf_hyperg_luke](#), 131
- [__conf_hyperg_series](#), 131
- [__coshint](#), 131
- [__cyl_bessel](#), 132
- [__cyl_bessel_i](#), 132
- [__cyl_bessel_ij_series](#), 133
- [__cyl_bessel_ik](#), 133
- [__cyl_bessel_ik_asymp](#), 134
- [__cyl_bessel_ik_steel](#), 134
- [__cyl_bessel_j](#), 134
- [__cyl_bessel_jn](#), 135
- [__cyl_bessel_jn_asymp](#), 135
- [__cyl_bessel_jn_steel](#), 135
- [__cyl_bessel_k](#), 136
- [__cyl_hankel_1](#), 136, 137
- [__cyl_hankel_2](#), 137
- [__cyl_neumann](#), 138
- [__cyl_neumann_n](#), 138
- [__dawson](#), 138
- [__dawson_const_frac](#), 139
- [__dawson_series](#), 139
- [__debye_region](#), 139
- [__dilog](#), 139
- [__dirichlet_beta](#), 140
- [__dirichlet_eta](#), 140, 141
- [__double_factorial](#), 141
- [__ellint_1](#), 141
- [__ellint_2](#), 142
- [__ellint_3](#), 142
- [__ellint_cel](#), 142
- [__ellint_d](#), 143
- [__ellint_el1](#), 143
- [__ellint_el2](#), 143
- [__ellint_el3](#), 143
- [__ellint_rc](#), 143
- [__ellint_rd](#), 144
- [__ellint_rf](#), 144
- [__ellint_rg](#), 145
- [__ellint_rj](#), 145
- [__ellnome](#), 146
- [__ellnome_k](#), 146
- [__ellnome_series](#), 146
- [__expint](#), 146, 147
- [__expint_E1](#), 147
- [__expint_E1_asymp](#), 148
- [__expint_E1_series](#), 148
- [__expint_Ei](#), 149
- [__expint_Ei_asymp](#), 149
- [__expint_Ei_series](#), 149
- [__expint_En_cont_frac](#), 150

- [__expint_En_recursion](#), 150
- [__expint_En_series](#), 151
- [__expint_asymp](#), 147
- [__expint_large_n](#), 151
- [__f_cdf](#), 151
- [__f_cdfc](#), 152
- [__factorial](#), 152
- [__fermi_dirac](#), 152
- [__fock_airy](#), 153
- [__fpequal](#), 153
- [__fpimag](#), 153, 155
- [__fpreal](#), 155
- [__fresnel](#), 155
- [__fresnel_cont_frac](#), 156
- [__fresnel_series](#), 156
- [__gamma](#), 156
- [__gamma_cont_frac](#), 156
- [__gamma_l](#), 156
- [__gamma_p](#), 157
- [__gamma_q](#), 157
- [__gamma_series](#), 157
- [__gamma_temme](#), 157
- [__gamma_u](#), 158
- [__gauss](#), 158
- [__gegenbauer_poly](#), 158
- [__hankel](#), 158
- [__hankel_debye](#), 159
- [__hankel_params](#), 159
- [__hankel_uniform](#), 159
- [__hankel_uniform_olver](#), 160
- [__hankel_uniform_outer](#), 160
- [__hankel_uniform_sum](#), 160
- [__heuman_lambda](#), 161
- [__hurwitz_zeta](#), 161
- [__hurwitz_zeta_euler_maclaurin](#), 162
- [__hydrogen](#), 162
- [__hyperg](#), 162
- [__hyperg_luke](#), 163
- [__hyperg_reflect](#), 163
- [__hyperg_series](#), 163
- [__jacobi_sncndn](#), 164
- [__jacobi_zeta](#), 164
- [__laguerre](#), 164
- [__log_bincoef](#), 164
- [__log_double_factorial](#), 165
- [__log_factorial](#), 165
- [__log_gamma](#), 165
- [__log_gamma_bernoulli](#), 166
- [__log_gamma_lanczos](#), 166
- [__log_gamma_sign](#), 166
- [__log_gamma_spouge](#), 167
- [__log_pochhammer_l](#), 167
- [__log_pochhammer_u](#), 168
- [__logint](#), 168
- [__owens_t](#), 168
- [__pochhammer_l](#), 169
- [__pochhammer_u](#), 169
- [__poly_hermite](#), 169
- [__poly_hermite_asymp](#), 170
- [__poly_hermite_recursion](#), 170
- [__poly_jacobi](#), 170
- [__poly_laguerre](#), 171
- [__poly_laguerre_hyperg](#), 171
- [__poly_laguerre_large_n](#), 173
- [__poly_laguerre_recursion](#), 173
- [__poly_legendre_p](#), 174
- [__poly_legendre_q](#), 174
- [__poly_radial_jacobi](#), 174
- [__polylog](#), 174, 176
- [__polylog_exp](#), 176
- [__polylog_exp_asymp](#), 176
- [__polylog_exp_int_neg](#), 177
- [__polylog_exp_int_pos](#), 178
- [__polylog_exp_neg](#), 178, 179
- [__polylog_exp_neg_even](#), 179
- [__polylog_exp_neg_odd](#), 180
- [__polylog_exp_negative_real_part](#), 181
- [__polylog_exp_pos](#), 181, 182
- [__polylog_exp_real_neg](#), 183
- [__polylog_exp_real_pos](#), 183, 184
- [__psi](#), 184
- [__psi_asymp](#), 185
- [__psi_series](#), 185
- [__riemann_zeta](#), 185
- [__riemann_zeta_alt](#), 186
- [__riemann_zeta_euler_maclaurin](#), 186
- [__riemann_zeta_glob](#), 186
- [__riemann_zeta_m_1](#), 186
- [__riemann_zeta_m_1_sum](#), 188
- [__riemann_zeta_product](#), 188
- [__riemann_zeta_sum](#), 188
- [__sinc](#), 189
- [__sinc_pi](#), 189
- [__sincosint](#), 189
- [__sincosint_asymp](#), 189
- [__sincosint_cont_frac](#), 190
- [__sincosint_series](#), 190
- [__sinhc](#), 190
- [__sinhc_pi](#), 190
- [__sinhint](#), 190
- [__sph_bessel](#), 191
- [__sph_bessel_ik](#), 191
- [__sph_bessel_jn](#), 193
- [__sph_hankel](#), 193
- [__sph_hankel_1](#), 193, 194
- [__sph_hankel_2](#), 194, 195
- [__sph_harmonic](#), 195
- [__sph_legendre](#), 195

- __sph_neumann, [196](#)
- __students_t_cdf, [196](#)
- __students_t_cdfc, [197](#)
- __theta_1, [197](#)
- __theta_2, [197](#)
- __theta_2_asyp, [198](#)
- __theta_2_sum, [198](#)
- __theta_3, [198](#)
- __theta_3_asyp, [198](#)
- __theta_3_sum, [199](#)
- __theta_4, [199](#)
- __theta_c, [199](#)
- __theta_d, [199](#)
- __theta_n, [199](#)
- __theta_s, [200](#)
- __zernike, [200](#)
- __znorm1, [200](#)
- __znorm2, [200](#)
- COSINT, [112](#)
- evenzeta, [200](#)
- SININT, [112](#)
- std::__detail::_Factorial_table
 - __factorial, [203](#)
 - __log_factorial, [203](#)
 - __n, [203](#)
- std::__detail::_Factorial_table<_Tp>, [203](#)
- theta_1
 - Extended Mathematical Special Functions, [61](#)
- theta_1f
 - Extended Mathematical Special Functions, [61](#)
- theta_1l
 - Extended Mathematical Special Functions, [61](#)
- theta_2
 - Extended Mathematical Special Functions, [61](#)
- theta_2f
 - Extended Mathematical Special Functions, [62](#)
- theta_2l
 - Extended Mathematical Special Functions, [62](#)
- theta_3
 - Extended Mathematical Special Functions, [62](#)
- theta_3f
 - Extended Mathematical Special Functions, [62](#)
- theta_3l
 - Extended Mathematical Special Functions, [62](#)
- theta_4
 - Extended Mathematical Special Functions, [63](#)
- theta_4f
 - Extended Mathematical Special Functions, [63](#)
- theta_4l
 - Extended Mathematical Special Functions, [63](#)
- theta_c
 - Extended Mathematical Special Functions, [63](#)
- theta_cf
 - Extended Mathematical Special Functions, [64](#)
- theta_cl
 - Extended Mathematical Special Functions, [64](#)
- theta_d
 - Extended Mathematical Special Functions, [64](#)
- theta_df
 - Extended Mathematical Special Functions, [64](#)
- theta_dl
 - Extended Mathematical Special Functions, [64](#)
- theta_n
 - Extended Mathematical Special Functions, [65](#)
- theta_nf
 - Extended Mathematical Special Functions, [65](#)
- theta_nl
 - Extended Mathematical Special Functions, [65](#)
- theta_s
 - Extended Mathematical Special Functions, [65](#)
- theta_sf
 - Extended Mathematical Special Functions, [66](#)
- theta_sl
 - Extended Mathematical Special Functions, [66](#)
- zernike
 - Extended Mathematical Special Functions, [66](#)
- zernikef
 - Extended Mathematical Special Functions, [66](#)
- zernikel
 - Extended Mathematical Special Functions, [66](#)