

# O2scl - An Object-Oriented Scientific Computing Library

Version 0.8

# Contents

<b>1</b>	<b>O2scl User's Guide</b>	<b>1</b>
1.1	Quick Reference to User's Guide . . . . .	2
1.2	Motivation . . . . .	3
1.3	Installation . . . . .	3
1.4	General Usage . . . . .	4
1.5	Complex Numbers . . . . .	5
1.6	Vectors and Matrices . . . . .	5
1.7	Interpolation . . . . .	11
1.8	Physical constants . . . . .	11
1.9	Function Objects . . . . .	11
1.10	Data tables . . . . .	13
1.11	Differentiation . . . . .	15
1.12	Integration . . . . .	15
1.13	Roots of Polynomials . . . . .	16
1.14	Equation Solving . . . . .	16
1.15	Minimization . . . . .	17
1.16	Monte Carlo Integration . . . . .	17
1.17	Simulated Annealing . . . . .	17
1.18	Non-linear Least-Squares Fitting . . . . .	17
1.19	Solution of Ordinary Differential Equations . . . . .	17
1.20	Random Number Generation . . . . .	17
1.21	Two-dimensional Interpolation . . . . .	18
1.22	Other Routines . . . . .	18
1.23	Library settings . . . . .	18
1.24	Object I/O . . . . .	18
1.25	Other Examples . . . . .	21
1.26	Design Considerations . . . . .	24
1.27	License Information . . . . .	26
1.28	Acknowledgements . . . . .	42
1.29	Bibliography . . . . .	42
<b>2</b>	<b>O2scl Namespace Documentation</b>	<b>42</b>
2.1	gsl_cgs Namespace Reference . . . . .	42
2.2	gsl_cgsm Namespace Reference . . . . .	46
2.3	gsl_mks Namespace Reference . . . . .	50
2.4	gsl_mkksa Namespace Reference . . . . .	53
2.5	gsl_num Namespace Reference . . . . .	57

2.6	o2scl_arith Namespace Reference	58
2.7	o2scl_const Namespace Reference	59
2.8	o2scl_fm Namespace Reference	61
2.9	o2scl_inte_qag_coeffs Namespace Reference	62
2.10	o2scl_inte_qng_coeffs Namespace Reference	65
<b>3</b>	<b>O2scl Data Structure Documentation</b>	<b>66</b>
3.1	adapt_step Class Template Reference	66
3.2	array_2d_alloc Class Template Reference	68
3.3	array_alloc Class Template Reference	68
3.4	array_const_reverse Class Template Reference	69
3.5	array_const_subvector Class Reference	69
3.6	array_const_subvector_reverse Class Reference	70
3.7	array_interp Class Template Reference	71
3.8	array_interp_vec Class Template Reference	71
3.9	array_reverse Class Template Reference	72
3.10	array_row Class Template Reference	72
3.11	array_subvector Class Reference	73
3.12	array_subvector_reverse Class Reference	73
3.13	base_interp Class Template Reference	74
3.14	base_ioc Class Reference	75
3.15	bin_size Class Reference	76
3.16	binary_in_file Class Reference	77
3.17	binary_out_file Class Reference	78
3.18	bool_io_type Class Reference	80
3.19	cern_adapt Class Template Reference	80
3.20	cern_cauchy Class Template Reference	82
3.21	cern_cubic_real_coeff Class Reference	83
3.22	cern_deriv Class Template Reference	84
3.23	cern_gauss Class Template Reference	85
3.24	cern_gauss56 Class Template Reference	87
3.25	cern_minimize Class Template Reference	87
3.26	cern_mroot Class Template Reference	89
3.27	cern_mroot_root Class Template Reference	91
3.28	cern_quartic_real_coeff Class Reference	94
3.29	cern_root Class Template Reference	94
3.30	char_io_type Class Reference	96
3.31	cinput Class Reference	97

3.32 cli Class Reference . . . . .	98
3.33 cmd_line_arg Struct Reference . . . . .	100
3.34 col Struct Reference . . . . .	101
3.35 collection Class Reference . . . . .	101
3.36 collection::iterator Class Reference . . . . .	105
3.37 collection::type_iterator Class Reference . . . . .	106
3.38 collection_entry Struct Reference . . . . .	107
3.39 columnify Class Reference . . . . .	108
3.40 comm_option Class Reference . . . . .	109
3.41 comm_option_funct Class Reference . . . . .	109
3.42 comm_option_mfptr Class Template Reference . . . . .	110
3.43 comm_option_s Struct Reference . . . . .	111
3.44 comp_gen_inte Class Template Reference . . . . .	112
3.45 comp_gen_inte::od_parms Struct Reference . . . . .	113
3.46 composite_inte Class Template Reference . . . . .	114
3.47 composite_inte::od_parms Struct Reference . . . . .	115
3.48 contour Class Reference . . . . .	116
3.49 coutput Class Reference . . . . .	122
3.50 coutput::ltptr Struct Reference . . . . .	124
3.51 cspline_interp Class Template Reference . . . . .	124
3.52 cubic_complex Class Reference . . . . .	125
3.53 cubic_real Class Reference . . . . .	126
3.54 cubic_real_coeff Class Reference . . . . .	127
3.55 cubic_std_complex Class Reference . . . . .	128
3.56 deriv Class Template Reference . . . . .	129
3.57 deriv::dpars Struct Reference . . . . .	131
3.58 deriv_ioc Class Reference . . . . .	131
3.59 double_io_type Class Reference . . . . .	132
3.60 eqi_deriv Class Template Reference . . . . .	132
3.61 err_class Class Reference . . . . .	135
3.62 exact_jacobian Class Template Reference . . . . .	137
3.63 exact_jacobian::ej_parms Struct Reference . . . . .	138
3.64 file_detect Class Reference . . . . .	139
3.65 fit_base Class Template Reference . . . . .	140
3.66 fit_fix_pars Class Template Reference . . . . .	141
3.67 fit_funct Class Template Reference . . . . .	143
3.68 fit_funct_fptr Class Template Reference . . . . .	143
3.69 fit_funct_mfptr Class Template Reference . . . . .	144

---

3.70	<a href="#">fit_vfunct Class Template Reference</a>	145
3.71	<a href="#">fit_vfunct_fptr Class Template Reference</a>	146
3.72	<a href="#">fit_vfunct_mfptr Class Template Reference</a>	147
3.73	<a href="#">funct Class Template Reference</a>	147
3.74	<a href="#">funct_fptr Class Template Reference</a>	148
3.75	<a href="#">funct_fptr_noerr Class Template Reference</a>	149
3.76	<a href="#">funct_fptr_nopar Class Template Reference</a>	150
3.77	<a href="#">funct_mfptr Class Template Reference</a>	150
3.78	<a href="#">funct_mfptr_noerr Class Template Reference</a>	151
3.79	<a href="#">funct_mfptr_nopar Class Template Reference</a>	152
3.80	<a href="#">gaussian_2d Class Template Reference</a>	153
3.81	<a href="#">gen_inte Class Template Reference</a>	153
3.82	<a href="#">gen_test_number Class Template Reference</a>	154
3.83	<a href="#">gm_parms Struct Template Reference</a>	155
3.84	<a href="#">gm_parms Struct Template Reference</a>	156
3.85	<a href="#">gm_parms Struct Template Reference</a>	156
3.86	<a href="#">grad_funct Class Template Reference</a>	157
3.87	<a href="#">grad_funct_fptr Class Template Reference</a>	157
3.88	<a href="#">grad_funct_mfptr Class Template Reference</a>	158
3.89	<a href="#">grad_vfunct Class Template Reference</a>	159
3.90	<a href="#">grad_vfunct_fptr Class Template Reference</a>	159
3.91	<a href="#">grad_vfunct_mfptr Class Template Reference</a>	160
3.92	<a href="#">gradient Class Template Reference</a>	161
3.93	<a href="#">gradient_array Class Template Reference</a>	161
3.94	<a href="#">gsl_anneal Class Template Reference</a>	162
3.95	<a href="#">gsl_astep Class Template Reference</a>	163
3.96	<a href="#">gsl_astep::gsl_ode_control Struct Reference</a>	165
3.97	<a href="#">gsl_astep::gsl_odeiv_evolve Class Reference</a>	166
3.98	<a href="#">gsl_chebapp Class Template Reference</a>	166
3.99	<a href="#">gsl_cubic_real_coeff Class Reference</a>	168
3.100	<a href="#">gsl_deriv Class Template Reference</a>	169
3.101	<a href="#">gsl_fft Class Reference</a>	170
3.102	<a href="#">gsl_fit Class Template Reference</a>	171
3.103	<a href="#">gsl_fit::func_par Struct Reference</a>	173
3.104	<a href="#">gsl_HH_solver Class Reference</a>	174
3.105	<a href="#">gsl_inte Class Reference</a>	174
3.106	<a href="#">gsl_inte_cheb Class Template Reference</a>	176
3.107	<a href="#">gsl_inte_kronrod Class Template Reference</a>	176

---

3.108gsl_inte_qag Class Template Reference . . . . .	177
3.109gsl_inte_qagi Class Template Reference . . . . .	179
3.110gsl_inte_qagil Class Template Reference . . . . .	180
3.111gsl_inte_qagiu Class Template Reference . . . . .	181
3.112gsl_inte_qags Class Template Reference . . . . .	183
3.113gsl_inte_qawc Class Template Reference . . . . .	184
3.114gsl_inte_qawf_cos Class Template Reference . . . . .	185
3.115gsl_inte_qawf_sin Class Template Reference . . . . .	186
3.116gsl_inte_qawo_cos Class Template Reference . . . . .	187
3.117gsl_inte_qawo_sin Class Template Reference . . . . .	188
3.118gsl_inte_qaws Class Template Reference . . . . .	190
3.119gsl_inte_qaws::fn_qaws_params Struct Reference . . . . .	191
3.120gsl_inte_qng Class Template Reference . . . . .	192
3.121gsl_inte_singular Class Template Reference . . . . .	193
3.122gsl_inte_singular::extrapolation_table Struct Reference . . . . .	194
3.123gsl_inte_table Class Reference . . . . .	194
3.124gsl_inte_transform Class Template Reference . . . . .	196
3.125gsl_LU_solver Class Reference . . . . .	197
3.126gsl_min_brent Class Template Reference . . . . .	197
3.127gsl_miser Class Template Reference . . . . .	199
3.128gsl_mmin_base Class Template Reference . . . . .	200
3.129gsl_mmin_bfgs2 Class Template Reference . . . . .	202
3.130gsl_mmin_conf Class Template Reference . . . . .	203
3.131gsl_mmin_conf_array Class Template Reference . . . . .	206
3.132gsl_mmin_conp Class Template Reference . . . . .	206
3.133gsl_mmin_linmin Class Reference . . . . .	207
3.134gsl_mmin_simp Class Template Reference . . . . .	208
3.135gsl_mmin_simp_b Class Reference . . . . .	211
3.136gsl_mmin_simp_b::simp_state_t Struct Reference . . . . .	212
3.137gsl_mmin_wrap_base Class Reference . . . . .	212
3.138gsl_mmin_wrapper Class Template Reference . . . . .	213
3.139gsl_monte Class Template Reference . . . . .	214
3.140gsl_mroot_hybrids Class Template Reference . . . . .	215
3.141gsl_mroot_hybrids::o2scl_hybrid_state_t Struct Reference . . . . .	218
3.142gsl_poly_real_coeff Class Reference . . . . .	218
3.143gsl_QR_solver Class Reference . . . . .	220
3.144gsl_quadratic_real_coeff Class Reference . . . . .	220
3.145gsl_quartic_real Class Reference . . . . .	221

---

3.146gsl_quartic_real2 Class Reference . . . . .	222
3.147gsl_rk8pd Class Template Reference . . . . .	222
3.148gsl_rk8pd_fast Class Template Reference . . . . .	224
3.149gsl_rkck Class Template Reference . . . . .	226
3.150gsl_rkck_fast Class Template Reference . . . . .	227
3.151gsl_rnga Class Reference . . . . .	228
3.152gsl_root_brent Class Template Reference . . . . .	229
3.153gsl_root_stef Class Template Reference . . . . .	231
3.154gsl_series Class Reference . . . . .	233
3.155gsl_vegas Class Template Reference . . . . .	233
3.156hybrids_base Class Reference . . . . .	234
3.157in_file_format Class Reference . . . . .	235
3.158int_io_type Class Reference . . . . .	236
3.159inte Class Template Reference . . . . .	237
3.160io_base Class Reference . . . . .	238
3.161io_manager Class Reference . . . . .	241
3.162io_tlate Class Template Reference . . . . .	242
3.163io_type_info Class Reference . . . . .	246
3.164io_vtlate Class Template Reference . . . . .	247
3.165jac_funct Class Template Reference . . . . .	248
3.166jac_funct_fptr Class Template Reference . . . . .	249
3.167jac_funct_mfptr Class Template Reference . . . . .	249
3.168jacobian Class Template Reference . . . . .	250
3.169lanczos Class Template Reference . . . . .	251
3.170lib_settings_class Class Reference . . . . .	252
3.171linear_interp Class Template Reference . . . . .	253
3.172linear_solver Class Template Reference . . . . .	254
3.173long_io_type Class Reference . . . . .	254
3.174mcarlo_inte Class Template Reference . . . . .	255
3.175min_fit Class Template Reference . . . . .	256
3.176min_fit::func_par Struct Reference . . . . .	257
3.177minimize Class Template Reference . . . . .	258
3.178mm_funct Class Template Reference . . . . .	260
3.179mm_funct_fptr Class Template Reference . . . . .	261
3.180mm_funct_fptr_nopar Class Template Reference . . . . .	261
3.181mm_funct_gsl Class Template Reference . . . . .	262
3.182mm_funct_mfptr Class Template Reference . . . . .	263
3.183mm_funct_mfptr_nopar Class Template Reference . . . . .	264

---

3.184mm_vfunct Class Template Reference . . . . .	265
3.185mm_vfunct_fptr Class Template Reference . . . . .	266
3.186mm_vfunct_fptr_nopar Class Template Reference . . . . .	266
3.187mm_vfunct_gsl Class Template Reference . . . . .	267
3.188mm_vfunct_mfptr Class Template Reference . . . . .	268
3.189mm_vfunct_mfptr_nopar Class Template Reference . . . . .	269
3.190mroot Class Template Reference . . . . .	270
3.191multi_funct Class Template Reference . . . . .	272
3.192multi_funct_fptr Class Template Reference . . . . .	273
3.193multi_funct_fptr_noerr Class Template Reference . . . . .	274
3.194multi_funct_gsl Class Template Reference . . . . .	275
3.195multi_funct_mfptr Class Template Reference . . . . .	276
3.196multi_funct_mfptr_noerr Class Template Reference . . . . .	277
3.197multi_inte Class Template Reference . . . . .	278
3.198multi_min Class Template Reference . . . . .	279
3.199multi_min_fix Class Template Reference . . . . .	280
3.200multi_vfunct Class Template Reference . . . . .	282
3.201multi_vfunct_fptr Class Template Reference . . . . .	282
3.202multi_vfunct_fptr_noerr Class Template Reference . . . . .	284
3.203multi_vfunct_gsl Class Template Reference . . . . .	285
3.204multi_vfunct_mfptr Class Template Reference . . . . .	286
3.205multi_vfunct_mfptr_noerr Class Template Reference . . . . .	287
3.206naive_metropolis Class Template Reference . . . . .	288
3.207naive_quartic_complex Class Reference . . . . .	290
3.208naive_quartic_real Class Reference . . . . .	291
3.209nonadapt_step Class Template Reference . . . . .	292
3.210o2scl_interp Class Template Reference . . . . .	293
3.211o2scl_interp_vec Class Template Reference . . . . .	294
3.212ode_bv_shoot Class Template Reference . . . . .	295
3.213ode_bv_solve Class Template Reference . . . . .	296
3.214ode_funct Class Template Reference . . . . .	297
3.215ode_funct_fptr Class Template Reference . . . . .	298
3.216ode_funct_mfptr Class Template Reference . . . . .	299
3.217ode_it_funct Class Template Reference . . . . .	300
3.218ode_it_funct_fptr Class Template Reference . . . . .	300
3.219ode_it_funct_mfptr Class Template Reference . . . . .	301
3.220ode_it_make_Coord Class Reference . . . . .	302
3.221ode_it_solve Class Template Reference . . . . .	303

---



3.222ode_iv_solve Class Template Reference . . . . .	304
3.223ode_vfunct Class Template Reference . . . . .	305
3.224ode_vfunct_fptr Class Template Reference . . . . .	306
3.225ode_vfunct_mfptr Class Template Reference . . . . .	307
3.226odestep Class Template Reference . . . . .	308
3.227ofmatrix Class Template Reference . . . . .	309
3.228ofvector Class Template Reference . . . . .	309
3.229ofvector_cx Class Template Reference . . . . .	310
3.230omatrix_alloc Class Reference . . . . .	311
3.231omatrix_col_tlate Class Template Reference . . . . .	311
3.232omatrix_const_col_tlate Class Template Reference . . . . .	312
3.233omatrix_const_row_tlate Class Template Reference . . . . .	312
3.234omatrix_cx_col_tlate Class Template Reference . . . . .	313
3.235omatrix_cx_const_col_tlate Class Template Reference . . . . .	313
3.236omatrix_cx_const_row_tlate Class Template Reference . . . . .	314
3.237omatrix_cx_row_tlate Class Template Reference . . . . .	314
3.238omatrix_cx_tlate Class Template Reference . . . . .	315
3.239omatrix_cx_view_tlate Class Template Reference . . . . .	316
3.240omatrix_diag_tlate Class Template Reference . . . . .	318
3.241omatrix_row_tlate Class Template Reference . . . . .	319
3.242omatrix_tlate Class Template Reference . . . . .	319
3.243omatrix_view_tlate Class Template Reference . . . . .	321
3.244other_ioc Class Reference . . . . .	323
3.245other_todos_and_bugs Class Reference . . . . .	323
3.246out_file_format Class Reference . . . . .	324
3.247ovector_alloc Class Reference . . . . .	325
3.248ovector_array_stride_tlate Class Template Reference . . . . .	325
3.249ovector_array_tlate Class Template Reference . . . . .	326
3.250ovector_const_array_stride_tlate Class Template Reference . . . . .	326
3.251ovector_const_array_tlate Class Template Reference . . . . .	327
3.252ovector_const_reverse_tlate Class Template Reference . . . . .	328
3.253ovector_const_subvector_reverse_tlate Class Template Reference . . . . .	329
3.254ovector_const_subvector_tlate Class Template Reference . . . . .	330
3.255ovector_cx_array_stride_tlate Class Template Reference . . . . .	332
3.256ovector_cx_array_tlate Class Template Reference . . . . .	332
3.257ovector_cx_const_array_stride_tlate Class Template Reference . . . . .	333
3.258ovector_cx_const_array_tlate Class Template Reference . . . . .	334
3.259ovector_cx_const_subvector_tlate Class Template Reference . . . . .	335

---

3.260ovector_cx_imag_tlate Class Template Reference . . . . .	336
3.261ovector_cx_real_tlate Class Template Reference . . . . .	336
3.262ovector_cx_subvector_tlate Class Template Reference . . . . .	337
3.263ovector_cx_tlate Class Template Reference . . . . .	337
3.264ovector_cx_view_tlate Class Template Reference . . . . .	339
3.265ovector_int_alloc Class Reference . . . . .	341
3.266ovector_reverse_tlate Class Template Reference . . . . .	342
3.267ovector_subvector_reverse_tlate Class Template Reference . . . . .	343
3.268ovector_subvector_tlate Class Template Reference . . . . .	344
3.269ovector_tlate Class Template Reference . . . . .	344
3.270ovector_view_tlate Class Template Reference . . . . .	346
3.271pinside Class Reference . . . . .	349
3.272pinside::line Struct Reference . . . . .	350
3.273pinside::point Struct Reference . . . . .	350
3.274planar_intp Class Template Reference . . . . .	351
3.275pointer_alloc Class Template Reference . . . . .	352
3.276pointer_input Struct Reference . . . . .	352
3.277pointer_output Struct Reference . . . . .	353
3.278poly_complex Class Reference . . . . .	353
3.279poly_real_coeff Class Reference . . . . .	354
3.280polylog Class Reference . . . . .	355
3.281quad_intp Class Template Reference . . . . .	356
3.282quad_intp::point Struct Reference . . . . .	358
3.283quadratic_complex Class Reference . . . . .	358
3.284quadratic_real Class Reference . . . . .	359
3.285quadratic_real_coeff Class Reference . . . . .	360
3.286quadratic_std_complex Class Reference . . . . .	360
3.287quartic_complex Class Reference . . . . .	361
3.288quartic_real Class Reference . . . . .	362
3.289quartic_real_coeff Class Reference . . . . .	363
3.290rnga Class Reference . . . . .	364
3.291root Class Template Reference . . . . .	365
3.292search_vec Class Template Reference . . . . .	367
3.293sim_anneal Class Template Reference . . . . .	368
3.294simple_grad Class Template Reference . . . . .	370
3.295simple_grad_array Class Template Reference . . . . .	370
3.296simple_jacobian Class Template Reference . . . . .	371
3.297sma_interp Class Template Reference . . . . .	372

---

3.298sma_interp_vec Class Template Reference . . . . .	373
3.299smart_interp Class Template Reference . . . . .	373
3.300smart_interp_vec Class Template Reference . . . . .	375
3.301sortd Struct Reference . . . . .	376
3.302string_comp Struct Reference . . . . .	376
3.303string_io_type Class Reference . . . . .	377
3.304table Class Reference . . . . .	377
3.305test_mgr Class Reference . . . . .	387
3.306text_in_file Class Reference . . . . .	389
3.307text_out_file Class Reference . . . . .	390
3.308timer_clock Class Reference . . . . .	393
3.309timer_gettod Class Reference . . . . .	394
3.310tptr_geoseries Class Template Reference . . . . .	394
3.311tptr_schedule Class Template Reference . . . . .	395
3.312twod_eqi_intp Class Reference . . . . .	396
3.313twod_intp Class Reference . . . . .	397
3.314ufmatrix Class Template Reference . . . . .	399
3.315ufmatrix_cx Class Template Reference . . . . .	399
3.316ufvector Class Template Reference . . . . .	400
3.317umatrix_alloc Class Reference . . . . .	401
3.318umatrix_const_row_tlate Class Template Reference . . . . .	401
3.319umatrix_cx_alloc Class Reference . . . . .	402
3.320umatrix_cx_const_row_tlate Class Template Reference . . . . .	402
3.321umatrix_cx_row_tlate Class Template Reference . . . . .	403
3.322umatrix_cx_tlate Class Template Reference . . . . .	403
3.323umatrix_cx_view_tlate Class Template Reference . . . . .	405
3.324umatrix_row_tlate Class Template Reference . . . . .	407
3.325umatrix_tlate Class Template Reference . . . . .	407
3.326umatrix_view_tlate Class Template Reference . . . . .	409
3.327uvector_array_tlate Class Template Reference . . . . .	411
3.328uvector_const_array_tlate Class Template Reference . . . . .	411
3.329uvector_const_subvector_tlate Class Template Reference . . . . .	412
3.330uvector_cx_array_tlate Class Template Reference . . . . .	413
3.331uvector_cx_const_array_tlate Class Template Reference . . . . .	413
3.332uvector_cx_const_subvector_tlate Class Template Reference . . . . .	414
3.333uvector_cx_subvector_tlate Class Template Reference . . . . .	415
3.334uvector_cx_tlate Class Template Reference . . . . .	415
3.335uvector_cx_view_tlate Class Template Reference . . . . .	416

---

3.336	<a href="#">uvector_subvector_tlate Class Template Reference</a>	418
3.337	<a href="#">uvector_tlate Class Template Reference</a>	419
3.338	<a href="#">uvector_view_tlate Class Template Reference</a>	420
3.339	<a href="#">word_io_type Class Reference</a>	422
<b>4</b>	<b>O2scl File Documentation</b>	<b>423</b>
4.1	<a href="#">array.h File Reference</a>	423
4.2	<a href="#">columnify.h File Reference</a>	425
4.3	<a href="#">cx_arith.h File Reference</a>	426
4.4	<a href="#">err_hnd.h File Reference</a>	428
4.5	<a href="#">lib_settings.h File Reference</a>	431
4.6	<a href="#">minimize.h File Reference</a>	432
4.7	<a href="#">misc.h File Reference</a>	433
4.8	<a href="#">omatrix_cx_tlate.h File Reference</a>	435
4.9	<a href="#">omatrix_tlate.h File Reference</a>	436
4.10	<a href="#">ovector_cx_tlate.h File Reference</a>	438
4.11	<a href="#">ovector_tlate.h File Reference</a>	440
4.12	<a href="#">poly.h File Reference</a>	442
4.13	<a href="#">string_conv.h File Reference</a>	443
4.14	<a href="#">umatrix_cx_tlate.h File Reference</a>	445
4.15	<a href="#">umatrix_tlate.h File Reference</a>	447
4.16	<a href="#">uvector_cx_tlate.h File Reference</a>	448
4.17	<a href="#">uvector_tlate.h File Reference</a>	450
4.18	<a href="#">vec_arith.h File Reference</a>	451
<b>5</b>	<b>O2scl Page Documentation</b>	<b>454</b>
5.1	<a href="#">Pre-Subversion Change Log</a>	454
5.2	<a href="#">Todo List</a>	457
5.3	<a href="#">Download O2scl</a>	462
5.4	<a href="#">Ideas for future development</a>	463
5.5	<a href="#">Bug List</a>	464

## 1 O2scl User's Guide

O2scl is a C++ class library for object-oriented numerical programming. It includes

- Classes based on numerical routines from GSL and CERNLIB
- Vector and matrix classes which are fully compatible with `gsl_vector` and `gsl_matrix`, yet offer indexing with `operator[]` and other object-oriented features
- The CERNLIB-based classes are rewritten in C++ and are often faster than their GSL counterparts

- Classes which require function inputs are designed to accept (public or private) member functions, even if they are virtual.
- Classes use templated vector types, which allow the use of object-oriented vectors or C-style arrays. Because of this, the O2scl versions of GSL algorithms can be significantly faster than the original.
- Highly compatible - Recent versions have been tested on Linux (32- and 64-bit systems, with Intel and AMD chips), Windows XP with Cygwin, and MacOSX.
- Free! O2scl is provided under Version 3 of the Gnu Public License
- Two mini-libraries
  - Thermodynamics of ideal and nearly-ideal particles with quantum statistics
  - Equations of state for finite density relevant for neutron stars

See licensing information at [License Information](#).

This is a **pre- $\beta$  version**. While I have released it to anyone who finds it useful, there may still be some bugs lurking around. I use many of these classes myself frequently and most things have been tested.

---

## 1.1 Quick Reference to User's Guide

- User's guide
    - [Installation](#)
    - [General Usage](#)
    - [Complex Numbers](#)
    - [Vectors and Matrices](#)
    - [Interpolation](#)
    - [Physical constants](#)
    - [Function Objects](#)
    - [Data tables](#)
    - [Differentiation](#)
    - [Integration](#)
    - [Roots of Polynomials](#)
    - [Equation Solving](#)
    - [Minimization](#)
    - [Monte Carlo Integration](#)
    - [Simulated Annealing](#)
    - [Non-linear Least-Squares Fitting](#)
    - [Solution of Ordinary Differential Equations](#)
    - [Random Number Generation](#)
    - [Two-dimensional Interpolation](#)
    - [Other Routines](#)
    - [Library settings](#)
    - [Object I/O](#)
    - [Other Examples](#)
    - [Design Considerations](#)
    - [License Information](#)
-

- [Acknowledgements](#)
  - [Bibliography](#)
  - [Todo List](#)
  - [Bug List](#)
- 

## 1.2 Motivation

I wanted a library for numerical programming in C++. This immediately suggests the following question: Why C++? Part of the answer to this question is practical; I am much more familiar with C/C++ so learning a different language (e.g. Fortran) never seemed like it made much sense. A related question might be: Why not an interpreted language, like python? Answer: faster execution. Finally, why C++ and not C? Answer: The object-oriented approach arguably allows for simpler and faster development while not appreciably removing flexibility. The object-orientation also encourages library development. Furthermore, I find the syntactic simplicity provided by C++ to be very convient at times.

In regards to the GNU Scientific Library (GSL), GSL is not simple to use for C++ development because it makes it almost impossible to utilize member functions in those routines which take functions as inputs.

I have also allowed the classes to hide their memory allocation structure, even though the user is frequently allowed to specify when it is allocated or deallocated. In light of the philosophy of 'encapsulation' this makes sense, as the memory requirements for the classes are not generic. Users who don't know about the details of the algorithm need not know about the details of the memory requirements, and users who need to know about the details of the memory requirements most often also want to know the details about the algorithm.

---

## 1.3 Installation

The rules for installation are generally the same as that for other GNU libraries. The file `INSTALL` has some details on this procedure. Generally, you should be able to run `./configure` and then type `make` and `make install`. The documentation is automatically installed by `make install`.

After `make install`, you may test the library by `make o2scl-test`. This should output a summary which is reproduced in the file `test-summary.text`. I've tried to ensure that the testing programs can automatically find the installed libraries using the compiler flags

(omitted)

If this doesn't work, you may have to add the correct directory to the `LD_LIBRARY_PATH` environment variable.

This library requires GSL and is designed to work with GSL versions 1.9. Some classes may work with older versions of GSL, but this cannot be guaranteed.

Range-checking for vectors and matrices is performed similar to the GSL approach, and is turned off by default. You can enable range-checking by defining `GSL_RANGE_CHECK=1`, e.g. `/code CPPFLAGS="-DGSL_RANGE_CHECK=1" ./configure /endcode`

The separate libraries `o2scl_eos` and `o2scl_part` are installed by default. To disable these libraries, configure with `-disable-eoslib` or `-disable-partlib`. Note that `o2scl_eos` depends on `o2scl_part` so using `-disable-partlib` without `-disable-eoslib` may not work.

There are several warning flags that are useful when configuring and compiling with `g++`. (See the GSL documentation for an excellent discussion.) For running `configure`, I use something like `/code CFLAGS="" CXXFLAGS="" CPPFLAGS="-ansi -pedantic -Wall -W \ -Wconversion -Wno-unused -Wshadow -Wpointer-arith -Wcast-align \ -Wwrite-strings -fshort-enums -ggdb -O3 -DGSL_RANGE_CHECK=0 \ -DHAVE_INLINE -I/home/asteiner/install/include" \ LDFLAGS="-L/home/asteiner/install/lib" ./configure -C \ -enable-readline -prefix=/home/asteiner/install /endcode` with the references to the directory `/home/asteiner/install` in order to install somewhere in my user directory, and because my copy of GSL is installed there, rather than in `/usr/local`.

---

The documentation is generated with Doxygen. In principle, the documentation can be regenerated by the end-user, but this is not supported.

**Un-installation:** While there is no explicit "uninstall" procedure, there are only a couple places to check. Installation creates directories named `o2scl` in the `include`, `doc` and `shared files` directory (which default to `/usr/local/include`, `/usr/local/doc`, and `/usr/local/share`) which can be removed. Finally, all of the libraries are named with the prefix `libo2scl` and are created by default in `/usr/local/lib`. As configured with the settings above, the files are in `/home/asteiner/install/include/o2scl`, `/home/asteiner/install/lib`, `/home/asteiner/install/share/o2scl`, and `/home/asteiner/install/doc/o2scl`.

---

## 1.4 General Usage

### Namespaces

Most of the classes reside in the namespace `o2scl` (removed from the documentation). Numerical and physical constants (many of them based on the GSL constants) are placed in separate namespaces. There are a couple other (mostly internal) namespaces which are documented separately.

### Documentation conventions

In the following documentation, function parameters are denoted by `parameter`, except when used in mathematical formulas as in variable.

### Nomenclature

Classes directly derived from the GNU Scientific Library are preceeded by the prefix `gsl_` and classes derived from CERNLIB are preceeded by the prefix `cern_`.

### Error handling

Error handling is GSL-like with functions returning 0 for success and calling a GSL-like error handler. The error handler, `err_hnd` is a global pointer to an object of type `err_class`.

Errors can be set through the macros `set_err`, `set_err_ret`, `add_err`, and `add_err_ret`, which are defined in the file `err_hnd.h`.

Functionality similar to `assert()` is provided with the macro `err_assert`, which exits if its argument is non-zero, and `bool_assert` which exits if its argument is false.

Note that the library functions do not typically try to reset the error handler. For this reason the user may need to reset the error handler before calling functions which do not return an integer error value so that they can easily test to see if an error occurred, e.g. using `err_hnd::get_errno()`.

The default error handler can be replaced by simply assigning the address of a descendant of `err_class` to `err_hnd`.

### Library dependencies

All of the libraries need `libo2scl_base`, `libgsl`, and either `libgslcblas` or some externally-specified `libcblas` to operate. Other additional dependencies are listed below:

- `libo2scl_eos` needs `libo2scl_nuclei`, `libo2scl_part`, `libo2scl_other`, `libo2scl_minimize`, `libo2scl_inte`, and `libo2scl_root`
- `libo2scl_part` needs `libo2scl_other`, `libo2scl_inte`, and `libo2scl_root`.
- `libo2scl_nuclei` needs `libo2scl_part`

### Verbose parameter

The solving and minimization classes have a verbose parameter that controls screen output for the higher-level interfaces. Values of verbose greater than 0 call a `print_iter()` function which prints out the indendent variables(s), the function values(s), and two numbers which display the progress in convergence. The first number is a value which quantifies how much progress has been made

---

through a number greater than zero which decreases as progress is made. The second number quantifies the tolerance that indicates success.

---

## 1.5 Complex Numbers

Some rudimentary arithmetic operators for `gsl_complex` are defined in `arith.h`, but no constructor has been written. The object `gsl_complex` is still a struct, not a class. For example,

```
gsl_complex a={{1,2}}, b={{3,4}};
gsl_complex c=a+b;
cout << GSL_REAL(c) << " " << GSL_IMAG(C) << endl;
```

In case the user needs to convert between `gsl_complex` and `std::complex<double>`, two conversion functions [gsl\\_to\\_complex\(\)](#) and [complex\\_to\\_gsl\(\)](#) are provided in [ovector\\_cx\\_tlate.h](#).

---

## 1.6 Vectors and Matrices

Vectors and matrices are designed using the templates [ovector\\_tlate](#) and [omatrix\\_tlate](#), which are compatible with `gsl_vector` and `gsl_matrix`. The GSL linear algebra routines can be easily used with [ovector\\_view\\_tlate](#), [omatrix\\_view\\_tlate](#) and related objects. Vectors with unit stride are provided in [uvector\\_tlate](#).

There is also a moderate amount of compatibility between [ovector\\_tlate](#), [ovector\\_view\\_tlate](#), [uvector\\_tlate](#), `std::vector<>` and the vectors from MV++. They all offer

- A blank constructor to create an empty vector
- A constructor with a "size" argument (`size_t` or `int`) to create a vector with a specified size.
- `operator=()` - Copy constructor
- `operator[]` - Array-indexing
- `size()` - Get the "size" of the vector

[ovector\\_tlate](#) and [omatrix\\_tlate](#) also offer the syntactic simplicity of array-like indexing, which is easy to apply to vectors and matrices created with GSL.

The following sections primarily discuss the operation objects of type [ovector](#). The generalizations to objects of type [uvector](#), [omatrix](#), [ovector\\_cx](#), and [omatrix\\_cx](#) are straightforward.

### Views

Vector and matrix views are provided as parents of the vector and matrix classes which do not have methods for memory allocation. As in GSL, it is simple to "view" normal arrays stored as `double *'s` (see [ovector\\_array](#)), parts of vectors ([ovector\\_subvector](#)), and rows and columns of matrices ([omatrix\\_row](#) and [omatrix\\_col](#)). Several operations are defined, including addition, subtraction, and dot products.

### Typedefs

Several typedefs are used to give smaller names to often used templates. Vectors and matrices of double-precision numbers all begin with the prefixes [ovector](#) and [omatrix](#). Integer versions begin with the prefixes [ovector\\_int](#) and [omatrix\\_int](#). Complex versions have an additional `"_cx"`, e.g. [ovector\\_cx](#). See [ovector\\_tlate.h](#), [ovector\\_cx\\_tlate.h](#), [omatrix\\_tlate.h](#), and [omatrix\\_cx\\_tlate.h](#).

### Unit-stride vectors

The [uvector\\_tlate](#) objects are naturally somewhat faster albeit less flexible than their finite-stride counterparts. Conversion to GSL vectors and matrices is not trivial for [uvector\\_tlate](#) objects, but demands copying the entire vector. Vector views, operators, and the nomenclature is similar to that of [ovector](#).

---



These unit-stride vectors are very useful to help optimize the internal workings of functions that do not need a finite stride. For example, O2scl implementations of some GSL functions can be somewhat faster than their GSL counterparts.

### Compatibility with MV++

There is a slight difference between how this works in comparison to MV++. The function `allocate()` operates a little differently than `newsize()`, as it will feel free to allocate new memory when `owner` is false. This should not be an issue, however, since it is not possible to create an `ovector_tlate` with a value of `owner` equal to zero, unless the `ovector_tlate` class is overloaded improperly.

### Memory allocation

Memory for vectors can be allocated using `ovector::allocate()` and `ovector::free()`. Allocation can also be performed by the constructor, and the destructor automatically calls `free()` if necessary. In contrast to `gsl_vector_alloc()`, `ovector::allocate()` will call `ovector::free()`, if necessary, to free previously allocated space. Allocating memory does not clear the associated memory. You can use `ovector::set_all()` with an argument of `0.0` to clear a vector or matrix.

If the memory allocation fails, either in the constructor or in `allocate()`, then the error handler will be called, partially allocated memory will be freed, and the size will be reset to zero. You can test to see if the allocation succeeded using something like

```
const size_t n=10;
ovector x(10);
if (x.size()==0) cout << "Failed." << endl;
```

### Get and set

Vectors can be modified using `ovector::get()` and `ovector::set()` methods analogous to `gsl_vector_get()` and `gsl_vector_set()`, or they can be modified through `ovector::operator[]` (or `ovector::operator()`), e.g.

```
ovector a(4);
a.set(0,2.0);
a.set(1,3.0);
a[2]=4.0;
a[3]=2.0*a[1];
```

If you want to set all of the values in an `ovector` at the same time, then use `ovector::set_all()`.

### Range checking

Range checking is performed, as in GSL, for `ovector_view_tlate::get()` and `ovector_view_tlate::set()`. Range-checking for vectors and matrices is turned on by default. To turn range-checking off, configure with `CPPFLAGS="-DGSL_RANGE_CHECK=0"`

If `O2SCL_ARRAY_ABORT` is defined, then `exit()` will be called any time a `gsl_index` error is called, e.g. an out-of-bounds array access. The user may also modify this behavior by replacing the default error handler.

### So2scallow and deep copy

Copying vectors is performed according to what kind of object is on the left-hand side (LHS) of the equals sign. If the LHS is a view, then a so2scallow copy is performed, and if the LHS is a `ovector`, then a deep copy is performed. If an attempt is made to perform a deep copy onto a vector that has already been allocated, then that previously allocated memory is automatically freed. The user must be careful to ensure that information is not lost this way, even though no memory leak will occur.

### Vector and matrix arithmetic

While some operations are possible from `ovector_view_tlate` objects, some operations demand the ability to allocate a temporary objects and thus demand `ovector_tlate` objects.

Vector\_view unary operators:

- `vector_view += vector_view`
- `vector_view -= vector_view`
- `vector_view += scalar`
- `vector_view -= scalar`

- `vector_view *= scalar`
- `scalar = norm(vector_view)`

Matrix\_view unary operators:

- `matrix += matrix`
- `matrix -= matrix`
- `matrix += scalar`
- `matrix -= scalar`
- `matrix *= scalar`

For more complicated vector/matrix operations, the GSL BLAS routines can be used directly with `ovector` and `omatrix` objects.

These arithmetic operations succeed even with non-matching vector and matrix sizes. For example, adding a 3x3 matrix to a 4x4 matrix will result in a 3x3 matrix and the 7 outer elements of the 4x4 matrix are ignored.

### Converting to and from GSL forms

Because of the way `ovector` is constructed, you may use type conversion to convert to and from objects of type `gsl_vector`.

```
ovector a(2);
a[0]=1.0;
a[1]=2.0;
gsl_vector *g=(gsl_vector *)(&a);
cout << gsl_vector_get(g,0) << " " << gsl_vector_get(g,1) << endl;
```

Or,

```
gsl_vector *g=gsl_vector_alloc(2);
gsl_vector_set(0,1.0);
gsl_vector_set(1,2.0);
ovector &a=(ovector &)(*g);
cout << a[0] << " " << a[1] << endl;
```

This sort of type-casting is usually discouraged, but is permissible here. In particular, this will not generate "type-punning" warnings in later gcc versions because `ovector` is derived from `gsl_vector`. If this bothers your sensibilities, however, then you can use the following approach:

```
ovector a(2);
gsl_vector *g=a.get_gsl_vector();
```

The ease of converting between these two kind of objects makes it easy to use `gsl` functions on objects of type `ovector`, i.e.

```
ovector a(2);
a[0]=2.0;
a[1]=1.0;
gsl_vector_sort((gsl_vector *)(&a));
cout << a[0] << " " << a[1] << endl;
```

### Converting from STL form

To "view" a `std::vector<double>`, you can use `ovector_array`

```
std::vector<double> d;
d.push_back(1.0);
d.push_back(3.0);
ovector_array aa(d.size,&(d[0]));
cout << aa[0] << " " << aa[1] << endl;
```

However, you should note that if the memory for the `std::vector` is reallocated (for example because of a call to `push_back()`), then a previously created `ovector_view` will be incorrect.

### **push\_back() and pop() methods**

These two functions give a behavior similar to the corresponding methods for `std::vector<>`. This will work in O2scl classes, but may not be compatible with all of the GSL functions. This will break if the address of a `ovector_tlate` is given to a GSL function which accesses the `block->size` parameter instead of the `size` parameter of a `gsl_vector`. Please contact the author if you find a GSL function with this behavior.

### **Views**

Views are slightly different than in GSL in that they are now implemented as parent classes. The code

```
double x[2]={1.0,2.0};
gsl_vector_view_array v(2,x);
gsl_vector *g=&(v.vector);
gsl_vector_set(g,0,3.0);
cout << gsl_vector_get(g,0) << " " << gsl_vector_get(g,1) << endl;
```

could be replaced by

```
double x[2]={1.0,2.0};
ovector_array a(2,x);
a[0]=3.0;
cout << a[0] << " " << a[1] << endl;
```

### **Passing ovector parameters**

It is often best to pass an `ovector` as a const reference to an `ovector_view`, i.e.

```
void function(const ovector_view &a);
```

If the function may change the values in the `ovector`, then just leave out `const`

```
void function(ovector_view &a);
```

This way, you ensure that the function is not allowed to modify the memory for the vector argument.

If you intend for a function (rather than the user) to handle the memory allocation, then some care is necessary. The following code

```
class my_class {
  int afunction(ovector &a) {
    a.allocate(1);
    // do something with a
    return 0;
  }
};
```

is confusing because the user may have already allocated memory for `a`. To avoid this, you may want to ensure that the user sends an empty vector. For example,

```
class my_class {
  int afunction(ovector &a) {
    if (a.get_size()>0 && a.is_owner()==true) {
      set_err("Unallocated vector not sent.",1);
      return 1;
    } else {
      a.allocate(1);
      // do something with a
      return 0;
    }
  }
};
```

In lieu of this, it is often preferable to use a local variable for the storage and offer a `get ()` function,

```
class my_class {
protected:
    ovector a;
public:
    int afunction() {
        a.allocate(1);
        // do something with a
        return 0;
    }
    int get_result(const ovector_view &av) { return a; }
};
```

The `O2scl` classes run into this situation quite frequently, but the vector type is specified through a template

```
template<class vec_t> class my_class {
protected:
    vec_t a;
public:
    int afunction(vec_t &a) {
        a.allocate(1);
        // do something with a
        return 0;
    }
};
```

### Vectors and `operator=()`

An "`operator=(value)`" method for setting all vector elements to the same value is not included because it leads to confusion between, `ovector_tlate::operator=(const data_t &val)` and `ovector_tlate::ovector_tlate(size_t val)`. For example, after implementing `operator=()` and executing the following

```
ovector_int o1=2;
ovector_int o2;
o2=2;
```

`o1` will be a vector of size two, and `o2` will be an empty vector!

To set all of the vector elements to the same value, use `ovector_tlate::set_all()`. Because of the existence of constructors like `ovector_tlate::ovector_tlate(size_t val)`, the following code

```
ovector_int o1=2;
```

still compiles, and is equivalent to

```
ovector_int o1(2);
```

while the code

```
ovector_int o1;
o1=2;
```

will not compile. As a matter of style, `ovector_int o1(2);` is preferable to `ovector_int o1=2;` to avoid confusion.

### Matrix structure

The matrices are structured in exactly the same way as in `GSL`. For a matrix with 2 rows, 4 columns, and a "tda" or "trailing dimension" of 7, the memory for the matrix is structured in the following way:

```
00 01 02 03 XX XX XX
10 11 12 13 XX XX XX
```

where XX indicates portions of memory that are unused. The tda can be accessed through, for example, the method `omatrix_view_tlate::tda()`. The `get(size_t, size_t)` methods always take the row index as the first argument and the column index as the second argument.

### Reversing the order of vectors

You can get a reversed vector view from `ovector_reverse_tlate`, or `uvector_reverse_tlate`. For these classes, `operator[]` and related methods are redefined to perform the reversal. If you want to make many calls to these indexing methods for a reversed vector, then simply copying the vector to a reversed version may be faster.

### Const-correctness with vectors

There are several classes named with `"_const"` to provide different kinds of const views of const vectors. The keyword `const` still ought to be included to ensure that the object is treated properly. For example,

```
ovector o(2);
o[0]=3.0;
o[1]=-1.0;
const ovector_const_subvector ocs(o,1,1);
```

At present, const-correctness in O2scl can be improperly removed, for example, with

```
ovector o(2);
o[0]=3.0;
o[1]=-1.0;
ovector_const_subvector ocs(o,1,1);
ovector_view ov(ocs);
ov[0]=2.0;
```

### List of vector classes

- [ovector\\_tlate](#)
- [ovector\\_view\\_tlate](#)
- [ovector\\_array\\_tlate](#)
- [ovector\\_array\\_stride\\_tlate](#)
- [ovector\\_subvector\\_tlate](#)
- [ovector\\_const\\_array\\_tlate](#)
- [ovector\\_const\\_subvector\\_tlate](#)
- [ovector\\_reverse\\_tlate](#)
- [ovector\\_const\\_reverse\\_tlate](#)
- [ovector\\_subvector\\_reverse\\_tlate](#)
- [ovector\\_const\\_subvector\\_reverse\\_tlate](#)
- [ofvector](#)
- [ovector\\_alloc](#)

### Todo

Finish this list

---



---

## 1.7 Interpolation

The classes `smart_interp` and `smart_interp_vec` allow interpolation, lookup, differentiation, and integration of data given in two ovector. A mechanism is implemented to allow these procedures to succeed for functions which are non-monotonic or multiply-valued outside the region of interest. In contrast to the GSL routines, data which is presented with a decreasing independent variable is handled automatically.

### Lookup and binary search

The class `search_vec` contains a searching functions for objects of type `ovector` which are monotonic. Note that if you want to find the index of an `ovector` where a particular value is located without any assumptions with regard to the ordering, you can use `ovector::lookup()` which performs an exhaustive search.

### The two interfaces

There are two classes, `smart_interp` and `smart_interp_vec`, analogous to the difference between using `gsl_interp_eval()` and `gsl_spline_eval()` in GSL. You can create a `smart_interp` object and use it to interpolate among any pair of chosen vectors, i.e.

```
ovector x(20), y(20);
// fill x and y with data
smart_interp gi;
double y_o2sclf=gi.interp(0.5,20,x,y);
```

Alternatively, you can create a `smart_interp_vec` object which can be optimized for a pair of vectors that you specify in advance

```
ovector x(20), y(20);
// fill x and y with data
smart_interp_vec gi(20,x,y);
double y_o2sclf=gi.interp(0.5);
```

## 1.8 Physical constants

The constants from GSL are reworked with the type `const double` and placed in namespaces called `gsl_cgs`, `gsl_cgsm`, `gsl_mks`, `gsl_mkssa`, and `gsl_num`. Some additional constants are given in the namespace `o2scl_const`.

## 1.9 Function Objects

Functions are passed to numerical routines using template-based function classes. There are several basic "kinds" of function objects:

- `funct` : One function of one variable
- `multi_funct` : One function of several variables
- `mm_funct` : *n* functions of *n* variables
- `fit_funct` : One function of one variable with *n* fitting parameters
- `ode_funct` : *n* derivatives as a function of *n* function values and the value of the independent variable

The class name suffixes denote children of a generic function type which are created using different kinds of inputs:

- `_fptr`: function pointer for a static or global function
- `_gsl`: GSL-like function pointer

- `_mfptr`: function pointer template for a class member function
- `_strings`: functions specified using strings, e.g. `"x^2-2"`
- `_noerr`: (for `funct` and `multi_funct`) a function which directly returns the function value rather than returning an integer error value
- `_nopar`: (for `funct`) a function which has no parameters specified by a `void *` and directly returns the function value.

There is a small overhead associated with the indirection: a "user" accesses the function class which then calls function which was specified in the constructor of the function class. This overhead can always be avoided by inheriting directly from the function class and thus the user will make a direct call. The present framework is specifically designed not to require inheriting from a function class. In many problems, the overhead associated with the indirection is small. In problems where the associated overhead is not small, there are usually other optimization issues (such as error handling and bounds checking in the "user" of the function object) which are still larger.

### An example

`ex_fptr` gives an example of the how member functions and external parameters are supplied to numerical routines. In this case, a member function (which contains the equation  $-\sin(x - 0.2)/|x + 0.01| + c_1 + c_2 = 0$  where  $c_1$  is a private data member and  $c_2$  is an externally-supplied parameter, is passed to the `gsl_root_brent` routine, which solves the equation.

### ex\_fptr.cpp

```
#include <o2scl/funct.h>
#include <o2scl/gsl_root_brent.h>

using namespace std;
using namespace o2scl;

class my_class {
private:
    double parameter;

public:
    void set_parameter() { parameter=0.1; }

    // A function demonstrating the different ways of implementing
    // function parameters
    double function_to_solve(double x, double &p) {
        return (-sin(x-0.2))/(fabs(x+0.01))+parameter+p;
    }
};

int main(void) {

    cout.setf(ios::scientific);

    // The solver, specifying the type of the parameter (double)
    // and the function type (funct<double>)
    gsl_root_brent<double,funct<double> > solver;

    my_class c;
    c.set_parameter();

    // This is the magic that allows specification of class member
    // functions as functions to solve. This object-oriented approach
    // avoids the use of static variables and functions and multiple
    // inheritance at the expense of a little overhead. We need to
    // provide the address of an instantiated object and the address of
    // the member function.
    funct_mfptr_noerr<my_class,double> function(&c,&my_class::function_to_solve);
```

```
double x1=-10.0;
double x2=1.4;
double p=0.1;

// The value verbose=1 prints out iteration information
// and verbose=2 requires a keypress between iterations.
// The parameter p=0.1 is used.
solver.verbose=1;
solver.solve_bkt(x1,x2,p,function);

// This is actually a somewhat difficult function to solve because
// of the sinusoidal behavior.
cout << "Solution: " << x1
      << " Function value: " << c.function_to_solve(x1,p) << endl;

return 0;
}
```

The output is `ex_fptr.scr`:

```
gsl_root_brent Iteration: 1
-7.493339e+00  3.319097e-01  8.893339e+00  1.000000e-12
gsl_root_brent Iteration: 2
-3.046669e+00  1.654610e-01  4.446669e+00  1.000000e-12
gsl_root_brent Iteration: 3
-8.233347e-01  1.249812e+00  2.223335e+00  1.000000e-12
gsl_root_brent Iteration: 4
 8.008747e-01 -4.972275e-01  1.624209e+00  1.000000e-12
gsl_root_brent Iteration: 5
-1.123003e-02  1.706528e+02  8.121047e-01  1.000000e-12
gsl_root_brent Iteration: 6
 7.985153e-01 -4.968532e-01  8.097454e-01  1.000000e-12
gsl_root_brent Iteration: 7
 3.936426e-01 -2.767453e-01  4.048727e-01  1.000000e-12
gsl_root_brent Iteration: 8
 1.912063e-01  2.437043e-01  2.024363e-01  1.000000e-12
gsl_root_brent Iteration: 9
 2.859986e-01 -9.017918e-02  9.479229e-02  1.000000e-12
gsl_root_brent Iteration: 10
 2.603960e-01 -2.322546e-02  6.918967e-02  1.000000e-12
gsl_root_brent Iteration: 11
 2.522875e-01  7.390184e-04  8.108494e-03  1.000000e-12
gsl_root_brent Iteration: 12
 2.525375e-01 -2.232677e-05  2.500503e-04  1.000000e-12
gsl_root_brent Iteration: 13
 2.525302e-01 -2.081953e-08  2.427175e-04  1.000000e-12
gsl_root_brent Iteration: 14
 2.525302e-01  6.106227e-16  6.843994e-09  1.000000e-12
gsl_root_brent Iteration: 15
 2.525302e-01 -5.551115e-17  2.220446e-16  1.000000e-12
Solution: 2.525302e-01 Function value: -5.551115e-17
```

## 1.10 Data tables

The class `table` is a container to hold and perform operations on related columns of data. It supports column operations, interpolation, column reference by either name or index, binary searching (in the case of ordered columns), sorting, and fitting two columns to a user-specified function.

Also supplied is a command-line utility which performs various `table` manipulations from a file containing a `table` object to `std::cout`. The output format can be controlled through several command-line options. Column operations, and sorting of the output are also supported.

An example - `ex_table.cpp`



```

#include <iostream>
#include <o2scl/base_ioc.h>
#include <o2scl/table.h>
#include <o2scl/collection.h>
#include <o2scl/constants.h>

using namespace std;
using namespace o2scl;

int main(void) {

    cout.setf(ios::scientific);

    // Create a table with two columns. It is usually best to
    // specify in the constructor, the number of rows you will
    // need before hand, i.e. table dat(100), but we do not
    // do so here for demonstration purposes.
    table dat;
    dat.line_of_names("x y");

    for(int i=0;i<11;i++) {
        dat.set("x",i,((double)i)/10.0);
        dat.set("y",i,sin(dat.get("x",i)));
    }

    // We demonstrate interpolation and differentiation and
    // compare with the exact answers
    cout << " Result          Expected" << endl;
    cout << "-----" << endl;
    cout << " " << dat.interp("y",0.5,"x") << " " << asin(0.5) << endl;
    cout << " " << dat.deriv("x",0.5,"y") << " " << cos(0.5) << endl;
    cout << dat.deriv2("x",0.5,"y") << " " << -sin(0.5) << endl;
    cout << endl;

    // We can also create columns from algebraic formulas.
    dat.make_cols("cx=sqrt(1.0-sx*sx) sx=y");

    // We can add "constants" for use in formulas
    dat.add_const("pi",acos(-1.0));
    dat.make_col("sin(x*pi/2)","s2");

    // We get approximate derivatives for an entire column
    dat.deriv("x","sx","cx2");

    base_ioc bio;

    // We output this table to std::cout and also to a file
    collection co;
    cout.setf(ios::scientific);
    text_out_file tout(&cout);
    co.out_one(&tout,"table","dat",&dat);
    table_io.out_one("ex_table.dat","dat",&dat);

    return 0;
}

```

This produces the following output:

#### ex\_table.scr

```

Result          Expected
-----
5.236012e-01    5.235988e-01
8.775485e-01    8.775826e-01
-4.809884e-01   -4.794255e-01

table dat
79 n_cols 6 n_lines 11 1
pi 3.141593e+00

```

---

```

x y sx cx s2 cx2
0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00 0.000000e+00 9.999995e-01
1.000000e-01 9.983342e-02 9.983342e-02 9.950042e-01 1.564345e-01 9.950034e-01
2.000000e-01 1.986693e-01 1.986693e-01 9.800666e-01 3.090170e-01 9.800667e-01
3.000000e-01 2.955202e-01 2.955202e-01 9.553365e-01 4.539905e-01 9.553335e-01
4.000000e-01 3.894183e-01 3.894183e-01 9.210610e-01 5.877853e-01 9.210695e-01
5.000000e-01 4.794255e-01 4.794255e-01 8.775826e-01 7.071068e-01 8.775485e-01
6.000000e-01 5.646425e-01 5.646425e-01 8.253356e-01 8.090170e-01 8.254605e-01
7.000000e-01 6.442177e-01 6.442177e-01 7.648422e-01 8.910065e-01 7.643741e-01
8.000000e-01 7.173561e-01 7.173561e-01 6.967067e-01 9.510565e-01 6.984518e-01
9.000000e-01 7.833269e-01 7.833269e-01 6.216100e-01 9.876883e-01 6.150954e-01
1.000000e+00 8.414710e-01 8.414710e-01 5.403023e-01 1.000000e+00 5.646134e-01

```

---

## 1.11 Differentiation

Differentiation is performed by descendants of [deriv](#) and the classes are provided. These allow one to calculate either first, second, and third derivatives. The GSL approach is used in [gsl\\_deriv](#), and the CERNLIB routine is used in [cern\\_deriv](#). Both of these compute derivatives for a function specified using a descendant of [funct](#). For functions which are tabulated over equally-spaced abscissas, the class [eqi\\_deriv](#) is provided which applies the formulas from Abramowitz and Stegun at a specified order.

**Warning:** For [gsl\\_deriv](#) and [cern\\_deriv](#), the second and third derivatives are calculated by naive repeated application of the code for the first derivative and can be particularly troublesome if the function is not sufficiently smooth. Error estimation is also incorrect for second and third derivatives.

---

## 1.12 Integration

Integration is performed by descendants of [inte](#) and is provided in the library `o2scl_inte`.

There are several routines for one-dimensional integration.

- General integration over a finite interval: [cern\\_adapt](#), [cern\\_gauss](#), [cern\\_gauss56](#), [gsl\\_inte\\_qag](#), and [gsl\\_inte\\_qng](#).
- General integration from 0 to  $\infty$ : [gsl\\_inte\\_qagiu](#)
- General integration from  $-\infty$  to 0: [gsl\\_inte\\_qagil](#)
- General integration from  $-\infty$  to  $\infty$ : [gsl\\_inte\\_qagi](#)
- Integration for a finite interval over an function with equally spaced abscissas provided in an array: [eqi\\_inte](#)
- General integration over a finite interval for a function with singularities: [gsl\\_inte\\_qags](#) and [gsl\\_inte\\_qagp](#)
- Cauchy principal value integration over a finite interval: [cern\\_cauchy](#) and [gsl\\_inte\\_qawc](#)
- Integration over a function weighted by  $\cos(x)$  or  $\sin(x)$ : [gsl\\_inte\\_qawo](#)
- Fourier integrals: [gsl\\_inte\\_qawf](#).
- Integration over a weight function

$$W(x) = (x-a)^\alpha (b-x)^\beta \log^\mu(x-a) \log^\nu(b-x)$$

is performed by [gsl\\_inte\\_qaws](#).

### Note:

The variables [inte::tolx](#) and [inte::tolf](#) have the same role as the quantities usually denoted in the GSL integration routines by `epsabs` and `epsrel`.

---

Multi-dimensional hypercubic integration is performed by [composite\\_inte](#), the sole descendant of [multi\\_inte](#). [composite\\_inte](#) allows you to specify a set of one-dimensional integration routines (objects of type [inte](#)) and apply them to a multi-dimensional problem.

General multi-dimensional integration is performed by [comp\\_gen\\_inte](#), the sole descendant of [gen\\_inte](#). The user is allowed to specify a upper and lower limits which are functions of the variables for integrations which have not yet been performed, i.e. the n-dimensional integral

$$\int_{x_0=a_0}^{x_0=b_0} f(x_0) \int_{x_1=a_1(x_0)}^{x_1=b_1(x_0)} f(x_0, x_1) \dots \int_{x_{n-1}=a_{n-1}(x_0, x_1, \dots, x_{n-2})}^{x_{n-1}=b_{n-1}(x_0, x_1, \dots, x_{n-2})} f(x_0, x_1, \dots, x_{n-1}) dx_{n-1} \dots dx_1 dx_0$$

Again, one specifies a set of [inte](#) objects to apply to each variable to be integrated over.

Monte Carlo integration is also provided (see [Monte Carlo Integration](#)).

## 1.13 Roots of Polynomials

Classes are provided for solving quadratic, cubic, and quartic equations as well as general polynomials. There is a standard nomenclature: classes which handle polynomials with real coefficients and real roots end with the suffix "\_real" ([quadratic\\_real](#), [cubic\\_real](#) and [quartic\\_real](#)), classes which handle real coefficients and complex roots end with the suffix "\_real\_coeff" ([quadratic\\_real\\_coeff](#), [cubic\\_real\\_coeff](#), [quartic\\_real\\_coeff](#), and [poly\\_real\\_coeff](#)), and classes which handle complex polynomials with complex coefficients ([quadratic\\_complex](#), [cubic\\_complex](#), [quartic\\_complex](#), and [poly\\_complex](#)). Most of these routines do not separately handle cases where the leading coefficient is zero.

At present, the polynomial routines work with complex numbers as objects of type `std::complex<double>` and are located in library `o2scl_other`.

For quadratics, [gsl\\_quadratic\\_real\\_coeff](#) is the best if the coefficients are real, while if the coefficients are complex, [quadratic\\_std\\_complex](#) is the only option at present. For cubics with real coefficients, [cern\\_cubic\\_real\\_coeff](#) is the best, while if the coefficients are complex, use [cubic\\_std\\_complex](#).

For a quartic polynomial with real coefficients, [cern\\_quartic\\_real\\_coeff](#) is the best, unless the coefficients of odd powers happen to be small, in which case, [gsl\\_quartic\\_real](#) works better. For quartics, generic polynomial solvers such as [gsl\\_poly\\_real\\_coeff](#) can provide more accurate (but slower) results. If the coefficients are complex, then you can use [naive\\_quartic\\_complex](#).

## 1.14 Equation Solving

Equation solving classes are stored in the library `o2scl_root`.

Solution of one equation in one variable is accomplished by children of the class [root](#). This base class provides the structure for three different solving methods:

- `root::solve(double &x, void *pa, funct &func)` which solves a function given an initial guess `x`
- `root::solve_bkt(double &x1, double x2, void *pa, funct &func)` which solves a function given a solution bracketed between `x1` and `x2`. The values of the function at `x1` and `x2` must have different signs.
- `root::solve_de(double &x, void *pa, funct &func, funct &df)` which solves a function given an initial guess `x` and the derivative of the function `df`.

If not all of these three functions are overloaded, then the source code in [root](#) is designed to try to automatically provide the solution using the remaining functions. Most of the one-dimensional solving routines, in their original form, are written in the second or third form above. For example, [gsl\\_root\\_brent](#) is originally a bracketing routine of the form `root::solve_bkt()`, but calls to either `root::solve()` or `root::solve_de()` will attempt to automatically bracket the function given the initial guess that is provided. Also, [gsl\\_root\\_stef](#) is a "root-polishing" routine given derivatives of the form `root::solve_de()`. If either `root::solve()` or `root::solve_bkt()` are called, then `root::solve_de()` will be called with finite-differencing used to estimate the derivative. Of course, it is frequently most efficient to use the solver in the way it was intended.

Solution of more than one equation is accomplished by descendants of the class [mroot](#). There are two basic functions

- `mroot::msolve(size_t n, ovector &x, void *pa, mm_func &func)` which solves the `n` equations given in `func` with an initial guess `x`.

All of the multi-dimensional methods can be used in one dimension, i.e. calling `mroot::solve()` will use the multi-dimensional method to solve a one-dimensional function.

---

## 1.15 Minimization

One-dimensional minimization is performed by descendants of [minimize](#) and provided in the library `o2scl_minimize`. One-dimensional minimization algorithms can be either of the bracketing type, where an interval and an initial guess must be provided, or a polishing algorithm where an initial guess is improved to produce a minimum.

Multi-dimensional minimization is performed by descendants of [multi\\_min](#). All of these multi-dimensional methods can also be used in one dimension. Simulated annealing methods are also provided (see [Simulated Annealing](#)).

It is important to note that not all of the minimization routines test the second derivative to ensure that it doesn't vanish to ensure that we have found a true minimum.

---

## 1.16 Monte Carlo Integration

Monte Carlo integration is performed by descendants of [mcarlo\\_inte](#) in the library `o2scl_mcarlo` ([gsl\\_monte](#), [gsl\\_miser](#), and [gsl\\_vegas](#))

These routines are generally superior to the direct methods for integrals over regions with large numbers of spatial dimensions.

---

## 1.17 Simulated Annealing

Minimization by simulated annealing is performed by descendants of [sim\\_anneal](#) (see [gsl\\_anneal](#)), given an annealing schedule given by descendants of [tptr\\_schedule](#) (see [tptr\\_geoseries](#)).

---

## 1.18 Non-linear Least-Squares Fitting

Fitting is performed by descendants of [fit\\_base](#) and fitting functions can be specified using [fit\\_func](#). The GSL fitting routines (scaled and unscaled) are implemented in [gsl\\_fit](#). A generic fitting routine using a minimizer object specified as a child of [multi\\_min](#) is implemented in [min\\_fit](#). When the [multi\\_min](#) object is (for example) a [sim\\_anneal](#) object, [min\\_fit](#) can avoid local minima which can occur when fitting noisy data.

---

## 1.19 Solution of Ordinary Differential Equations

Classes for non-adaptive integration are provided as descendants of [odestep](#) and classes for adaptive integration are descendants of [adapt\\_step](#).

Solution of initial value problems is performed by [ode\\_iv\\_solve](#), and boundary value problems by [ode\\_bv\\_solve](#).

---

## 1.20 Random Number Generation

Random number generators are descendants of [rnga](#) and are provided in the library `o2scl_rnga`. While the base object [rnga](#) is created to allow user-defined random number generators, the only random number generator presently included are from GSL. The

---

GSL random number generator code is reimplemented in the class [gsl\\_rng](#) to avoid an additional performance penalty. This is not a truly object-oriented interface in that it does not use virtual functions. This avoids any possible performance penalty. Random number generators are implemented as templates in [sim\\_anneal](#) and [mcarlo\\_inte](#).

---

## 1.21 Two-dimensional Interpolation

Successive use of [smart\\_interp](#) is implemented in [twod\\_intp](#). Also, see [planar\\_intp](#) and [quad\\_intp](#) and the computation of [contour](#) lines in [contour](#). These latter three classes are somewhat experimental at present.

---

## 1.22 Other Routines

**Fourier transforms** - see [gsl\\_fft](#)

**Series acceleration** - see [gsl\\_series](#)

**Chebyshev approximations** - see [gsl\\_chebapp](#)

**Timing execution** - see [timer\\_gettod](#) and [timer\\_clock](#)

**Polylogarithms** - see [polylog](#)

---

## 1.23 Library settings

There are a couple library settings which are handled by a global object [lib\\_settings](#) of type [lib\\_settings\\_class](#).

1. There are several data files that are used by various classes in the library. The installation procedure should ensure that these files are automatically found. However, if these data files are moved, then a call to [lib\\_settings\\_class::set\\_data\\_dir\(\)](#) can adjust the library to use the new directory. It is assumed that the directory structure within the directory has not changed.

---

## 1.24 Object I/O

The I/O portion of the library is still somewhat experimental.

Collections of objects can be stored in a [collection](#) class, and these collections can be written to or read from text or binary files. User-defined classes may be added to the collections and may be read and written to files as long as a descendant of [io\\_base](#) is provided.

Every type has an associated I/O type which is a descendant of [io\\_base](#). In order to perform any sort of input/output on any type, an object of the corresponding I/O type must be instantiated by the user. This is not done automatically by the library. (Since it doesn't know which objects are going to be used ahead of time, the library would have to instantiate *all* of the I/O objects, which is needlessly slow.) This makes the I/O slightly less user-friendly, but much more efficient. For convenience, each subsection of the library has a class (named with an `_ioc` suffix) which will automatically allocate all I/O types for that subsection.

**Level 1 functions:** Functions that input/output data from library-defined objects and internal types from files and combine these objects in collections. These are primarily member functions of the class [collection](#).

**Level 2 functions:** Functions which are designed to allow the user to input or output data for user-generated objects. These are primarily member functions of classes [cinput](#) and [coutput](#).

**Level 3 functions:** Functions which allow low-level modifications on how input and output is performed. Usage of level 3 functions is not immediately recommended for the casual user.

**Level 1 usage:**

For adding an object to a [collection](#) when you have a pointer to the I/O object for the associated type:

---

```
int collection::add(std::string name, io_base *tio, void *vec,
int sz=0, int sz2=0, bool overwrt=true, bool owner=false);
```

For adding an object to a [collection](#) otherwise:

```
int collection::add(std::string name, std::string stype,
void *vec, int sz=0, int sz2=0,
bool overwrt=true, bool owner=false);
```

To retrieve an object as a

```
void *
```

from a [collection](#) use one of:

```
int get(std::string tname, void *&vec);
int get(std::string tname, void *&vec, int &sz);
int get(std::string tname, void *&vec, int &sz, int &sz2);
int get(std::string tname, std::string &stype, void *&vec);
int get(std::string tname, std::string &stype, void *&vec, int &sz);
int get(std::string tname, std::string &stype, void *&vec, int &sz,
int &sz2);
```

When retrieving a scalar object without error- and type-checking you can use the shorthand version:

```
void *get(std::string name);
```

To output one object to a file:

```
int collection::out_one(out_file_format *outs, std::string stype,
std::string name, void *vp, int sz=0, int sz2=0);
```

To input one object from a file with a given type and name:

```
int collection::in_one_name(in_file_format *ins, std::string stype,
std::string name, void *&vp, int &sz, int &sz2);
```

To input the first object of a given type from a file:

```
int collection::in_one(in_file_format *ins, std::string stype,
std::string &name, void *&vp, int &sz, int &sz2);
```

### Level 2 usage (string-based):

If you don't have a pointer to the [io\\_base](#) child object corresponding to the type of subobject that you are manipulating, then you can use the following functions, which take the type name as a string.

To input a sub-object in an [io\\_base](#) template for which memory has already been allocated use one of:

```
int collection::object_in(std::string type, in_file_format *ins, void *vp,
std::string &name);
int collection::object_in(std::string type, in_file_format *ins, void *vp,
int sz, std::string &name);
int collection::object_in(std::string type, in_file_format *ins, void *vp,
int sz, int sz2, std::string &name);
```

To automatically allocate memory and input a sub-object of a [io\\_base](#) template use one of:

```
int collection::object_in_mem(std::string type, in_file_format *ins,
void *vp, std::string &name);
int collection::object_in_mem(std::string type, in_file_format *ins,
void *vp, int sz, std::string &name);
int collection::object_in_mem(std::string type, in_file_format *ins,
void *vp, int sz, int sz2, std::string &name);
```

To output a subobject in an [io\\_base](#) template use:

```
int collection::object_out(std::string type, out_file_format *outs,
void *op, int sz=0, int sz2=0, std::string name="");
```

### Level 2 usage (with [io\\_base](#) pointer):

To input a sub-object in an [io\\_base](#) template for which memory has already been allocated use one of:

```
virtual int object_in(cinput *cin, in_file_format *ins, object *op,
std::string &name);
virtual int object_in(cinput *cin, in_file_format *ins, object *op,
int sz, std::string &name);
virtual int object_in(cinput *cin, in_file_format *ins, object **op,
int sz, int sz2, std::string &name);
template<size_t N>
int object_in(cinput *co, in_file_format *ins,
object op[][N], int sz, std::string &name);
```

To automatically allocate memory and input a sub-object of a [io\\_base](#) template use one of:

```
virtual int object_in_mem(cinput *cin, in_file_format *ins,
object *&op, std::string &name);
virtual int object_in_mem(cinput *cin, in_file_format *ins, object *&op,
int &sz, std::string &name);
virtual int object_in_mem(cinput *cin, in_file_format *ins,
object **&op, int &sz, int &sz2,
std::string &name);
template<size_t N>
int object_in_mem(cinput *co, in_file_format *ins,
object op[][N], int &sz, std::string &name);
```

To output a subobject in an [io\\_base](#) template use:

```
virtual int object_out(coutput *cout, out_file_format *outs, object *op,
int sz=0, std::string name="");
virtual int object_out(coutput *cout, out_file_format *outs, object **op,
int sz, int sz2, std::string name="");
template<size_t N>
int object_out(coutput *cout, out_file_format *outs,
object op[][N], int sz, std::string name="");
```

To automatically allocate/deallocate memory for an object, use:

```
virtual int mem_alloc(object *&op);
virtual int mem_alloc_arr(object *&op, int sz);
virtual int mem_alloc_2darr(object **&op, int sz, int sz2);
virtual int mem_free(object *op);
virtual int mem_free_arr(object *op);
virtual int mem_free_2darr(object **op, int sz);
```

### Usage of [io\\_tlate](#)

The functions [io\\_tlate::input\(\)](#) and [io\\_tlate::output\(\)](#) need to be implemented for every class has information for I/O. For subobjects of the class, [cinput::object\\_in\(\)](#) and [cinput::object\\_out\(\)](#) can be called to input or output the information associated with the subobject.

For input, `cinut::object_in_name()`, `cinut::object_in_mem()`, and `cinut::object_in_mem_name()` allow the freedom to input an object with a name or with memory allocation. The function `coutut::object_out_name()` allows one to output an object with a name. If the class contains a pointer to the subobject, then `io_base::pointer_in()` or `io_base::pointer_out()` can be used.

---

## 1.25 Other Examples

All of the following examples are in the `examples` subdirectory.

### I/O example

This example demonstrates the ease of using text or binary files for input or output. A `collection` is filled with data and output to several files. The data is read in from each of the files and written to `cout`.

#### `ex_file.cpp`

```
#include <o2scl/collection.h>
#include <o2scl/file_detect.h>
#include <o2scl/base_ioc.h>

using namespace std;
using namespace o2scl;

int main(void) {
    cout.setf(ios::scientific);

    /// Some data
    char c='t';
    double d=sqrt(2.0);
    int i=10;
    string s="string test";

    /// Make sure the I/O objects are present
    base_ioc bio;

    /// We create a collection, and add all of the data
    /// items to the collection
    collection co;

    co.add("char_test", "char", &c);
    co.add("double_test", "double", &d);
    co.add("int_test", "int", &i);
    co.add("string_test", "string", &s);

    /// In order to output things to stdout later
    text_out_file tout(&cout);

    /// Output the data in the collection to several
    /// different types of files

    co.fout("ex_file_1");

    text_out_file *tof2=new text_out_file("ex_file_2.gz");
    co.fout(tof2);
    delete tof2;

    binary_out_file *bof=new binary_out_file("ex_file_3");
    co.fout(bof);
    delete bof;

    binary_out_file *bof2=new binary_out_file("ex_file_4.gz");
    co.fout(bof2);
    delete bof2;

    /// Erase all of the information in the collection
    co.clear();
```

---



```

/// Now read in each of the files

file_detect fd;

cout.precision(10);

cout << "ex_file_1:\n" << endl;
co.fin(fd.open("ex_file_1"));
if (fd.is_compressed()) cout << "Compressed." << endl;
if (fd.is_binary()) cout << "Binary." << endl;
fd.close();
co.fout(&tout);
co.clear();

cout << "\nex_file_2:\n" << endl;
co.fin(fd.open("ex_file_2.gz"));
if (fd.is_compressed()) cout << "Compressed." << endl;
if (fd.is_binary()) cout << "Binary." << endl;
fd.close();
co.fout(&tout);
co.clear();

cout << "\nex_file_3:\n" << endl;
co.fin(fd.open("ex_file_3"));
if (fd.is_compressed()) cout << "Compressed." << endl;
if (fd.is_binary()) cout << "Binary." << endl;
fd.close();
co.fout(&tout);
co.clear();

cout << "\nex_file_4:\n" << endl;
co.fin(fd.open("ex_file_4.gz"));
if (fd.is_compressed()) cout << "Compressed." << endl;
if (fd.is_binary()) cout << "Binary." << endl;
fd.close();
co.fout(&tout);
co.clear();

return 0;
}

```

This produces the following output:

#### **ex\_file.scr**

```

ex_file_1:

char char_test t
double double_test 1.4142100000e+00
int int_test 10
string string_test
string test

ex_file_2:

Compressed.
char char_test t
double double_test 1.4142100000e+00
int int_test 10
string string_test
string test

ex_file_3:

Binary.
char char_test t
double double_test 1.4142135624e+00
int int_test 10
string string_test

```

---

```

string test

ex_file_4:

Compressed.
Binary.
char char_test t
double double_test 1.4142135624e+00
int int_test 10
string string_test
string test

```

Note that the binary files automatically store the full precision for floating-point numbers, but that the text files do not. The precision for text files can be modified, see [text\\_out\\_file](#) for details.

## Differentiation

### ex\_diff.cpp

```

#include <iostream>

#include <o2scl/funct.h>
#include <o2scl/table.h>
#include <o2scl/eqi_deriv.h>
#include <o2scl/interp.h>
#include <o2scl/base_ioc.h>

// For cout
using namespace std;
// Most of the library resides in this namespace
using namespace o2scl;

int main(void) {
    size_t bign=30, i;

    cout.precision(6);
    cout.setf(ios::scientific);

    ovector x(bign), y(bign), dydx(bign), dydx2(bign);

    // We create some data, y=exp(x).
    for(i=0;i<bign;i++) {
        x[i]=((double)i)/3.0;
        y[i]=exp(x[i]);
    }

    // Compute dydx using derivative formulas for equally-spaced abscissas
    eqi_deriv<void *,funct<void *> > ei;
    ei.deriv_array(bign,1.0/3.0,y,dydx);

    // Compute the derivative using the interpolation
    // (natural cubic splines are the default)
    cspline_interp<ovector_view> it;
    o2scl_interp_vec<> gi(it,bign,x,y);
    for(i=0;i<bign;i++) {
        dydx2[i]=gi.deriv(x[i]);
    }

    // Create a data table
    table ta(100);
    // Give the column names
    ta.line_of_names("x y dydx dydx2 err err2");

    cout << "x          y          dydx          dydx' "
         << "          err          err' " << endl;
    for(i=0;i<bign;i++) {
        cout << x[i] << " " << y[i] << " " << dydx[i] << " " << dydx2[i] << " ";
        cout << fabs(y[i]-dydx[i])/y[i] << " " << fabs(y[i]-dydx2[i])/y[i] << endl;
        // Add the data to the table as well...
    }
}

```

```

double line[6]={x[i],y[i],dydx[i],dydx2[i],
               fabs(y[i]-dydx[i])/y[i],fabs(y[i]-dydx2[i])/y[i]};
ta.line_of_data(6,line);
}

// Create a collection, the main I/O facility
base_ioc bio;
collection co;
// Output results to file
co.out_one("ex_diff.out","table","ex_diff",&ta);

return 0;
}

```

The result of plotting `err` and `err'` from the output file `ex_diff.out` is

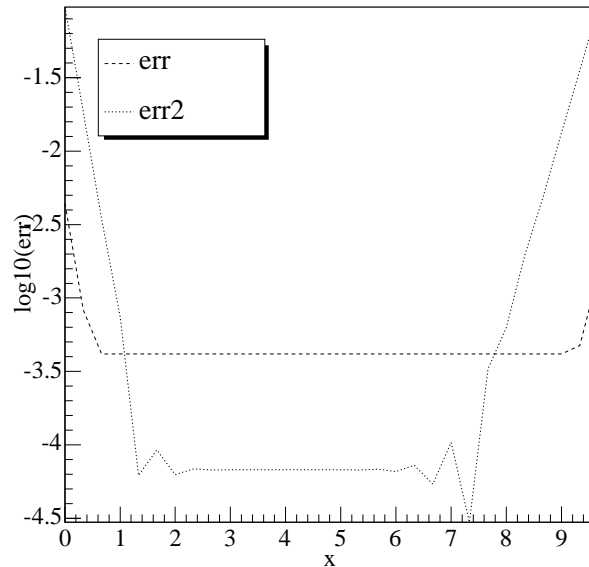


Figure 1: `ex_diff.out`

Note that the formula for equally-spaced abscissas provided by `eq_interval` is more accurate than interpolation with `smart_interp` at the boundaries, but less accurate otherwise.

## 1.26 Design Considerations

The design goal is to create an object-oriented computing library with classes that perform common numerical tasks. The most important principle is that the library should add functionality to the user while at the same time retaining as much freedom for the user as possible and allowing for ease of use and extensibility. To that end,

- The classes which utilize user-specified functions should be able to operate on member functions without requiring a particular inheritance structure,
- The interfaces ought to be generic so that the user can create new classes which perform related numerical tasks through inheritance,

- The classes should not use static variables or functions
- Const-correctness and type-safety should be used wherever possible, and
- The design should be somewhat compatible with GSL. Also, the library provides higher-level routines for situations which do not require lower-level access.

### Header file dependencies

For reference, it's useful to know how the top-level header files depend on each other, since it can be difficult to trace everything down. In the `base` directory, the following are the most "top-level" header files and their associated dependencies:

```
err_hnd.h : (none)
lib_settings.h : (none)
array.h: err_hnd.h
uvector_tlate.h: err_hnd.h
ovector_tlate.h: uvector_tlate.h array.h err_hnd.h
misc.h : ovector_tlate.h uvector_tlate.h array.h lib_settings.h err_hnd.h
test_mgr.h : misc.h ovector_tlate.h uvector_tlate.h
.          array.h lib_settings.h err_hnd.h
```

### Todo

Redo this section since we've added [string\\_conv.h](#)

### The use of templates

Templates are used for parameters, instead of using `void *`. This makes for longer compilation times so any code that can be removed conveniently from the header files should be put into source code files instead.

### Type-casting in vector and matrix design

O2scl uses a gsl-like approach where viewing `const double *` arrays is performed by explicitly casting away `const`'ness internally and then preventing the user from changing the data.

In `gsl-1.6`, the preprocessor output for `vector/view_source.c` is:

```
gsl_vector_const_view_array (const double * base, size_t n)
{
    _gsl_vector_const_view view = {{0, 0, 0, 0, 0}};

    if (n == 0)
    {
        do { gsl_error ("vector length n must be positive integer", "view_source.c", 28, GSL_EINVAL) ; return view ; } while (0)
    }
    {
        gsl_vector v = {0, 0, 0, 0, 0};

        v.data = (double *)base ;
        v.size = n;
        v.stride = 1;
        v.block = 0;
        v.owner = 0;
        ((_gsl_vector_view *)&view)->vector = v;

        return view;
    }
}
```

Note the explicit cast from `const double *` to `double *`. This is similar to what is done in `src/base/ovector.cpp`.

### Global objects

There are three global objects that are created in `libo2scl_base`:

- `def_err_hnd` is the default error handler
- `err_hnd` is the pointer to the error handler (points to `def_err_hnd` by default)
- `lib_settings` to control a few library settings

All other global objects are avoided.

### Thread safety

The classes are thread-safe, meaning that two instances of the same class will not clash if their methods are called concurrently since static variables are only used for compile-time constants. However, two threads cannot, in general, safely access the same instance of a class. In this respect, `O2scl` is no different from `GSL`.

### Documentation design

Throughout the header files, the command `\nothing` sometimes appears. This is an alias that doesn't do anything and is present just to simplify the tabification in `emacs`.

The commands `\comment` and `\endcomment` delineate comments about the documentation that are present in the header files but don't ever show up in the HTML or LaTeX documentation.

### Release Procedure

Internally, it's useful to keep track of the steps required to create a full release (under construction)

- `svn update`
- Update version numbers in `configure.ac`
- source `aconfig.scr`
- `make version-update`
- Edit version number in distribution link in [doc/o2scl/main.h](http://doc.o2scl/main.h)
- `make install`
- `make o2scl-test`
- Make the `ex_` and `bm_` files in the examples directory
- Make sure `acol.help` and `acol.usage` are up to date
- `make o2scl-doc`
- `make o2scl-docp`
- `make dist`
- `svn commit`

---

## 1.27 License Information

`O2scl` (as well as `CERNLIB` and the Gnu Scientific Library (`GSL`)) is licensed under the GPL as provided in the files `COPYING` and in `doc/o2scl/extras/gpl_license.txt`. After installation, it is included in the documentation in `PREFIX/doc/extras/gpl_license.txt` where the default `PREFIX` is `/usr/local`.

GNU GENERAL PUBLIC LICENSE  
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <<http://fsf.org/>>  
Everyone is permitted to copy and distribute verbatim copies

---

of this license document, but changing it is not allowed.

#### Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

#### TERMS AND CONDITIONS

##### 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

---

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

#### 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

---

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

## 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

## 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

## 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

## 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
-



b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

#### 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is

available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

#### 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of

it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

#### 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that

---

copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

#### 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

#### 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

#### 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a

---

party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

#### 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

#### 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

#### 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of

the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

#### 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

#### 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

#### How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>
```

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type 'show c' for details.
```

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

This documentation is provided under the GNU Free Documentation License, as given below and provided in `doc/o2scl/extras/fdl_license.txt`. After installation, it is included in the documentation in `PREFIX/doc/extras/fdl_license.txt` where the default PREFIX is `/usr/local`.

GNU Free Documentation License  
Version 1.2, November 2002

Copyright (C) 2000,2001,2002 Free Software Foundation, Inc.  
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA  
Everyone is permitted to copy and distribute verbatim copies  
of this license document, but changing it is not allowed.

## 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

## 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or



processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

## 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

## 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps,

when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

#### 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material

copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

## 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

## 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

## 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright

resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

#### 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

#### 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

#### 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

#### ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

---

```
Copyright (c)  YEAR  YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts,  
replace the "with...Texts." line with this:

```
with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other  
combination of the three, merge those two alternatives to suit the  
situation.

If your document contains nontrivial examples of program code, we  
recommend releasing these examples in parallel under your choice of  
free software license, such as the GNU General Public License,  
to permit their use in free software.

## 1.28 Acknowledgements

I would like to thank the creators of GSL for their excellent work

## 1.29 Bibliography

Some of the references which contain links should direct you to the work referred to directly through [dx.doi.org](https://dx.doi.org).

Bus75: J.C.P. Bus and T.J. Dekker, ACM Trans. Math. Software 1 (1975) 330-345.

Fletcher87: R. Fletcher, Practical methods of optimization (John Wiley & Sons, Chichester 1987) 39.

Krabs83: W. Krabs, Einführung in die lineare und nichtlineare Optimierung für Ingenieure (BSB B.G. Teubner, Leipzig 1983) 84.

Lewin83: L. Lewin, Polylogarithms and Associated Functions (North-Holland, New York, 1983).

Longman58: I.M. Longman, MTAC (later renamed Math. Comp.) **12** (1958) 205.

More79: J.J. More<sup>\*</sup> and M.Y. Cosnard, ACM Trans. Math. Software, **5** (1979) 64-85.

More80: J.J. More<sup>\*</sup> and M.Y. Cosnard, Algorithm 554 BRENTM, Collected Algorithms from CACM (1980).

Rutishauser63: H. Rutishauser, Ausdehnung des Rombergschen Prinzips (Extension of Romberg's Principle), Numer. Math. **5** (1963) 48-54.

## 2 O2scl Namespace Documentation

### 2.1 gsl\_cgs Namespace Reference

#### 2.1.1 Detailed Description

GSL constants in CGS units.

The CGS units are given below each constant

## Variables

- const double [schwarzschild\\_radius](#) = 2.95325008e5  
*cm*
- const double [speed\\_of\\_light](#) = 2.99792458e10  
*cm / s*
- const double [gravitational\\_constant](#) = 6.673e-8  
*cm<sup>3</sup> / g s<sup>2</sup>*
- const double [plancks\\_constant\\_h](#) = 6.62606876e-27  
*g cm<sup>2</sup> / s*
- const double [plancks\\_constant\\_hbar](#) = 1.05457159642e-27  
*g cm<sup>2</sup> / s*
- const double [astronomical\\_unit](#) = 1.49597870691e13  
*cm*
- const double [light\\_year](#) = 9.46053620707e17  
*cm*
- const double [parsec](#) = 3.08567758135e18  
*cm*
- const double [grav\\_accel](#) = 9.80665e2  
*cm / s<sup>2</sup>*
- const double [electron\\_volt](#) = 1.602176462e-12  
*g cm<sup>2</sup> / s<sup>2</sup>*
- const double [mass\\_electron](#) = 9.10938188e-28  
*g*
- const double [mass\\_muon](#) = 1.88353109e-25  
*g*
- const double [mass\\_proton](#) = 1.67262158e-24  
*g*
- const double [mass\\_neutron](#) = 1.67492716e-24  
*g*
- const double [rydberg](#) = 2.17987190389e-11  
*g cm<sup>2</sup> / s<sup>2</sup>*
- const double [boltzmann](#) = 1.3806503e-16  
*g cm<sup>2</sup> / K s<sup>2</sup>*
- const double [bohr\\_magneton](#) = 9.27400899e-20  
*A cm<sup>2</sup>.*
- const double [nuclear\\_magneton](#) = 5.05078317e-23  
*A cm<sup>2</sup>.*
- const double [electron\\_magnetic\\_moment](#) = 9.28476362e-20  
*A cm<sup>2</sup>.*
- const double [proton\\_magnetic\\_moment](#) = 1.410606633e-22  
*A cm<sup>2</sup>.*
- const double [molar\\_gas](#) = 8.314472e7  
*g cm<sup>2</sup> / K mol s<sup>2</sup>*
- const double [standard\\_gas\\_volume](#) = 2.2710981e4  
*cm<sup>3</sup> / mol*
- const double [minute](#) = 6e1  
*s*
- const double [hour](#) = 3.6e3  
*s*
- const double [day](#) = 8.64e4  
*s*
- const double [week](#) = 6.048e5  
*s*
- const double [inch](#) = 2.54e0  
*cm*
- const double [foot](#) = 3.048e1  
*cm*

- const double [yard](#) = 9.144e1  
*cm*
- const double [mile](#) = 1.609344e5  
*cm*
- const double [nautical\\_mile](#) = 1.852e5  
*cm*
- const double [fathom](#) = 1.8288e2  
*cm*
- const double [mil](#) = 2.54e-3  
*cm*
- const double [point](#) = 3.52777777778e-2  
*cm*
- const double [texpoint](#) = 3.51459803515e-2  
*cm*
- const double [micron](#) = 1e-4  
*cm*
- const double [angstrom](#) = 1e-8  
*cm*
- const double [hectare](#) = 1e8  
*cm*<sup>2</sup>
- const double [acre](#) = 4.04685642241e7  
*cm*<sup>2</sup>
- const double [barn](#) = 1e-24  
*cm*<sup>2</sup>
- const double [liter](#) = 1e3  
*cm*<sup>3</sup>
- const double [us\\_gallon](#) = 3.78541178402e3  
*cm*<sup>3</sup>
- const double [quart](#) = 9.46352946004e2  
*cm*<sup>3</sup>
- const double [pint](#) = 4.73176473002e2  
*cm*<sup>3</sup>
- const double [cup](#) = 2.36588236501e2  
*cm*<sup>3</sup>
- const double [fluid\\_ounce](#) = 2.95735295626e1  
*cm*<sup>3</sup>
- const double [tablespoon](#) = 1.47867647813e1  
*cm*<sup>3</sup>
- const double [teaspoon](#) = 4.92892159375e0  
*cm*<sup>3</sup>
- const double [canadian\\_gallon](#) = 4.54609e3  
*cm*<sup>3</sup>
- const double [uk\\_gallon](#) = 4.546092e3  
*cm*<sup>3</sup>
- const double [miles\\_per\\_hour](#) = 4.4704e1  
*cm* / *s*
- const double [kilometers\\_per\\_hour](#) = 2.77777777778e1  
*cm* / *s*
- const double [knot](#) = 5.14444444444e1  
*cm* / *s*
- const double [pound\\_mass](#) = 4.5359237e2  
*g*
- const double [ounce\\_mass](#) = 2.8349523125e1  
*g*
- const double [ton](#) = 9.0718474e5  
*g*
- const double [metric\\_ton](#) = 1e6  
*g*

- const double [uk\\_ton](#) = 1.0160469088e6  
*g*
- const double [troy\\_ounce](#) = 3.1103475e1  
*g*
- const double [carat](#) = 2e-1  
*g*
- const double [unified\\_atomic\\_mass](#) = 1.66053873e-24  
*g*
- const double [gram\\_force](#) = 9.80665e2  
*cm g / s^2*
- const double [pound\\_force](#) = 4.44822161526e5  
*cm g / s^2*
- const double [kilopound\\_force](#) = 4.44822161526e8  
*cm g / s^2*
- const double [poundal](#) = 1.38255e4  
*cm g / s^2*
- const double [calorie](#) = 4.1868e7  
*g cm^2 / s^2*
- const double [btu](#) = 1.05505585262e10  
*g cm^2 / s^2*
- const double [therm](#) = 1.05506e15  
*g cm^2 / s^2*
- const double [horsepower](#) = 7.457e9  
*g cm^2 / s^3*
- const double [bar](#) = 1e6  
*g / cm s^2*
- const double [std\\_atmosphere](#) = 1.01325e6  
*g / cm s^2*
- const double [torr](#) = 1.33322368421e3  
*g / cm s^2*
- const double [meter\\_of\\_mercury](#) = 1.33322368421e6  
*g / cm s^2*
- const double [inch\\_of\\_mercury](#) = 3.38638815789e4  
*g / cm s^2*
- const double [inch\\_of\\_water](#) = 2.490889e3  
*g / cm s^2*
- const double [psi](#) = 6.89475729317e4  
*g / cm s^2*
- const double [poise](#) = 1e0  
*g / cm s*
- const double [stokes](#) = 1e0  
*cm^2 / s*
- const double [faraday](#) = 9.6485341472e4  
*A s / mol.*
- const double [electron\\_charge](#) = 1.602176462e-19  
*A s.*
- const double [gauss](#) = 1e-1  
*g / A s^2*
- const double [stilb](#) = 1e0  
*cd / cm^2*
- const double [lumen](#) = 1e0  
*cd sr*
- const double [lux](#) = 1e-4  
*cd sr / cm^2*
- const double [phot](#) = 1e0  
*cd sr / cm^2*
- const double [footcandle](#) = 1.076e-3  
*cd sr / cm^2*



- const double [lambert](#) = 1e0  
*cd sr / cm<sup>2</sup>*
- const double [footlambert](#) = 1.07639104e-3  
*cd sr / cm<sup>2</sup>*
- const double [curie](#) = 3.7e10  
*I / s*
- const double [roentgen](#) = 2.58e-7  
*A s / g.*
- const double [rad](#) = 1e2  
*cm<sup>2</sup> / s<sup>2</sup>*
- const double [solar\\_mass](#) = 1.98892e33  
*g*
- const double [bohr\\_radius](#) = 5.291772083e-9  
*cm*
- const double [newton](#) = 1e5  
*cm g / s<sup>2</sup>*
- const double [dyne](#) = 1e0  
*cm g / s<sup>2</sup>*
- const double [joule](#) = 1e7  
*g cm<sup>2</sup> / s<sup>2</sup>*
- const double [erg](#) = 1e0  
*g cm<sup>2</sup> / s<sup>2</sup>*
- const double [stefan\\_boltzmann\\_constant](#) = 5.67039934436e-5  
*g / K<sup>4</sup> s<sup>3</sup>*
- const double [thomson\\_cross\\_section](#) = 6.65245853542e-25  
*cm<sup>2</sup>*

## 2.2 gsl\_cgsm Namespace Reference

### 2.2.1 Detailed Description

GSL constants in CGSM units.

The CGSM units are given below each constant

#### Variables

- const double [schwarzschild\\_radius](#) = 2.95325008e5  
*cm*
- const double [speed\\_of\\_light](#) = 2.99792458e10  
*cm / s*
- const double [gravitational\\_constant](#) = 6.673e-8  
*cm<sup>3</sup> / g s<sup>2</sup>*
- const double [plancks\\_constant\\_h](#) = 6.62606876e-27  
*g cm<sup>2</sup> / s*
- const double [plancks\\_constant\\_hbar](#) = 1.05457159642e-27  
*g cm<sup>2</sup> / s*
- const double [astronomical\\_unit](#) = 1.49597870691e13  
*cm*
- const double [light\\_year](#) = 9.46053620707e17  
*cm*
- const double [parsec](#) = 3.08567758135e18  
*cm*
- const double [grav\\_accel](#) = 9.80665e2  
*cm / s<sup>2</sup>*
- const double [electron\\_volt](#) = 1.602176462e-12  
*g cm<sup>2</sup> / s<sup>2</sup>*

- const double `mass_electron` = 9.10938188e-28  
*g*
- const double `mass_muon` = 1.88353109e-25  
*g*
- const double `mass_proton` = 1.67262158e-24  
*g*
- const double `mass_neutron` = 1.67492716e-24  
*g*
- const double `rydberg` = 2.17987190389e-11  
*g cm<sup>2</sup> / s<sup>2</sup>*
- const double `boltzmann` = 1.3806503e-16  
*g cm<sup>2</sup> / K s<sup>2</sup>*
- const double `bohr_magneton` = 9.27400899e-21  
*abamp cm<sup>2</sup>*
- const double `nuclear_magneton` = 5.05078317e-24  
*abamp cm<sup>2</sup>*
- const double `electron_magnetic_moment` = 9.28476362e-21  
*abamp cm<sup>2</sup>*
- const double `proton_magnetic_moment` = 1.410606633e-23  
*abamp cm<sup>2</sup>*
- const double `molar_gas` = 8.314472e7  
*g cm<sup>2</sup> / K mol s<sup>2</sup>*
- const double `standard_gas_volume` = 2.2710981e4  
*cm<sup>3</sup> / mol*
- const double `minute` = 6e1  
*s*
- const double `hour` = 3.6e3  
*s*
- const double `day` = 8.64e4  
*s*
- const double `week` = 6.048e5  
*s*
- const double `inch` = 2.54e0  
*cm*
- const double `foot` = 3.048e1  
*cm*
- const double `yard` = 9.144e1  
*cm*
- const double `mile` = 1.609344e5  
*cm*
- const double `nautical_mile` = 1.852e5  
*cm*
- const double `fathom` = 1.8288e2  
*cm*
- const double `mil` = 2.54e-3  
*cm*
- const double `point` = 3.52777777778e-2  
*cm*
- const double `texpoint` = 3.51459803515e-2  
*cm*
- const double `micron` = 1e-4  
*cm*
- const double `angstrom` = 1e-8  
*cm*
- const double `hectare` = 1e8  
*cm<sup>2</sup>*
- const double `acre` = 4.04685642241e7

- $cm^2$
  - const double [barn](#) = 1e-24
  - $cm^2$
  - const double [liter](#) = 1e3
  - $cm^3$
  - const double [us\\_gallon](#) = 3.78541178402e3
  - $cm^3$
  - const double [quart](#) = 9.46352946004e2
  - $cm^3$
  - const double [pint](#) = 4.73176473002e2
  - $cm^3$
  - const double [cup](#) = 2.36588236501e2
  - $cm^3$
  - const double [fluid\\_ounce](#) = 2.95735295626e1
  - $cm^3$
  - const double [tablespoon](#) = 1.47867647813e1
  - $cm^3$
  - const double [teaspoon](#) = 4.92892159375e0
  - $cm^3$
  - const double [canadian\\_gallon](#) = 4.54609e3
  - $cm^3$
  - const double [uk\\_gallon](#) = 4.546092e3
  - $cm^3$
  - const double [miles\\_per\\_hour](#) = 4.4704e1
  - $cm / s$
  - const double [kilometers\\_per\\_hour](#) = 2.77777777778e1
  - $cm / s$
  - const double [knot](#) = 5.14444444444e1
  - $cm / s$
  - const double [pound\\_mass](#) = 4.5359237e2
  - $g$
  - const double [ounce\\_mass](#) = 2.8349523125e1
  - $g$
  - const double [ton](#) = 9.0718474e5
  - $g$
  - const double [metric\\_ton](#) = 1e6
  - $g$
  - const double [uk\\_ton](#) = 1.0160469088e6
  - $g$
  - const double [troy\\_ounce](#) = 3.1103475e1
  - $g$
  - const double [carat](#) = 2e-1
  - $g$
  - const double [unified\\_atomic\\_mass](#) = 1.66053873e-24
  - $g$
  - const double [gram\\_force](#) = 9.80665e2
  - $cm\ g / s^2$
  - const double [pound\\_force](#) = 4.44822161526e5
  - $cm\ g / s^2$
  - const double [kilopound\\_force](#) = 4.44822161526e8
  - $cm\ g / s^2$
  - const double [poundal](#) = 1.38255e4
  - $cm\ g / s^2$
  - const double [calorie](#) = 4.1868e7
  - $g\ cm^2 / s^2$
  - const double [btu](#) = 1.05505585262e10
  - $g\ cm^2 / s^2$
  - const double [therm](#) = 1.05506e15
-

- $g\ cm^2 / s^2$
- const double [horsepower](#) = 7.457e9
- $g\ cm^2 / s^3$
- const double [bar](#) = 1e6
- $g / cm\ s^2$
- const double [std\\_atmosphere](#) = 1.01325e6
- $g / cm\ s^2$
- const double [torr](#) = 1.33322368421e3
- $g / cm\ s^2$
- const double [meter\\_of\\_mercury](#) = 1.33322368421e6
- $g / cm\ s^2$
- const double [inch\\_of\\_mercury](#) = 3.38638815789e4
- $g / cm\ s^2$
- const double [inch\\_of\\_water](#) = 2.490889e3
- $g / cm\ s^2$
- const double [psi](#) = 6.89475729317e4
- $g / cm\ s^2$
- const double [poise](#) = 1e0
- $g / cm\ s$
- const double [stokes](#) = 1e0
- $cm^2 / s$
- const double [faraday](#) = 9.6485341472e3
- $abamp\ s / mol$
- const double [electron\\_charge](#) = 1.602176462e-20
- $abamp\ s$
- const double [gauss](#) = 1e0
- $g / abamp\ s^2$
- const double [stilb](#) = 1e0
- $cd / cm^2$
- const double [lumen](#) = 1e0
- $cd\ sr$
- const double [lux](#) = 1e-4
- $cd\ sr / cm^2$
- const double [phot](#) = 1e0
- $cd\ sr / cm^2$
- const double [footcandle](#) = 1.076e-3
- $cd\ sr / cm^2$
- const double [lambert](#) = 1e0
- $cd\ sr / cm^2$
- const double [footlambert](#) = 1.07639104e-3
- $cd\ sr / cm^2$
- const double [curie](#) = 3.7e10
- $1 / s$
- const double [roentgen](#) = 2.58e-8
- $abamp\ s / g$
- const double [rad](#) = 1e2
- $cm^2 / s^2$
- const double [solar\\_mass](#) = 1.98892e33
- $g$
- const double [bohr\\_radius](#) = 5.291772083e-9
- $cm$
- const double [newton](#) = 1e5
- $cm\ g / s^2$
- const double [dyne](#) = 1e0
- $cm\ g / s^2$
- const double [joule](#) = 1e7
- $g\ cm^2 / s^2$
- const double [erg](#) = 1e0

$g \text{ cm}^2 / \text{s}^2$

- const double stefan\_boltzmann\_constant = 5.67039934436e-5  
 $g / \text{K}^4 \text{ s}^3$
- const double thomson\_cross\_section = 6.65245853542e-25  
 $\text{cm}^2$

## 2.3 gsl\_mks Namespace Reference

### 2.3.1 Detailed Description

GSL constants in MKS units.

The MKS units are given below each constant

#### Variables

- const double schwarzschild\_radius = 2.95325008e3  
 $m$
- const double speed\_of\_light = 2.99792458e8  
 $m / s$
- const double gravitational\_constant = 6.673e-11  
 $m^3 / kg \text{ s}^2$
- const double plancks\_constant\_h = 6.62606876e-34  
 $kg \text{ m}^2 / s$
- const double plancks\_constant\_hbar = 1.05457159642e-34  
 $kg \text{ m}^2 / s$
- const double astronomical\_unit = 1.49597870691e11  
 $m$
- const double light\_year = 9.46053620707e15  
 $m$
- const double parsec = 3.08567758135e16  
 $m$
- const double grav\_accel = 9.80665e0  
 $m / \text{s}^2$
- const double electron\_volt = 1.602176462e-19  
 $kg \text{ m}^2 / \text{s}^2$
- const double mass\_electron = 9.10938188e-31  
 $kg$
- const double mass\_muon = 1.88353109e-28  
 $kg$
- const double mass\_proton = 1.67262158e-27  
 $kg$
- const double mass\_neutron = 1.67492716e-27  
 $kg$
- const double rydberg = 2.17987190389e-18  
 $kg \text{ m}^2 / \text{s}^2$
- const double boltzmann = 1.3806503e-23  
 $kg \text{ m}^2 / \text{K s}^2$
- const double bohr\_magneton = 9.27400899e-24  
 $A \text{ m}^2$ .
- const double nuclear\_magneton = 5.05078317e-27  
 $A \text{ m}^2$ .
- const double electron\_magnetic\_moment = 9.28476362e-24  
 $A \text{ m}^2$ .
- const double proton\_magnetic\_moment = 1.410606633e-26  
 $A \text{ m}^2$ .

- const double `molar_gas` = 8.314472e0  
 $\text{kg m}^2 / \text{K mol s}^2$
  - const double `standard_gas_volume` = 2.2710981e-2  
 $\text{m}^3 / \text{mol}$
  - const double `minute` = 6e1  
 $\text{s}$
  - const double `hour` = 3.6e3  
 $\text{s}$
  - const double `day` = 8.64e4  
 $\text{s}$
  - const double `week` = 6.048e5  
 $\text{s}$
  - const double `inch` = 2.54e-2  
 $\text{m}$
  - const double `foot` = 3.048e-1  
 $\text{m}$
  - const double `yard` = 9.144e-1  
 $\text{m}$
  - const double `mile` = 1.609344e3  
 $\text{m}$
  - const double `nautical_mile` = 1.852e3  
 $\text{m}$
  - const double `fathom` = 1.8288e0  
 $\text{m}$
  - const double `mil` = 2.54e-5  
 $\text{m}$
  - const double `point` = 3.52777777778e-4  
 $\text{m}$
  - const double `texpoint` = 3.51459803515e-4  
 $\text{m}$
  - const double `micron` = 1e-6  
 $\text{m}$
  - const double `angstrom` = 1e-10  
 $\text{m}$
  - const double `hectare` = 1e4  
 $\text{m}^2$
  - const double `acre` = 4.04685642241e3  
 $\text{m}^2$
  - const double `barn` = 1e-28  
 $\text{m}^2$
  - const double `liter` = 1e-3  
 $\text{m}^3$
  - const double `us_gallon` = 3.78541178402e-3  
 $\text{m}^3$
  - const double `quart` = 9.46352946004e-4  
 $\text{m}^3$
  - const double `pint` = 4.73176473002e-4  
 $\text{m}^3$
  - const double `cup` = 2.36588236501e-4  
 $\text{m}^3$
  - const double `fluid_ounce` = 2.95735295626e-5  
 $\text{m}^3$
  - const double `tablespoon` = 1.47867647813e-5  
 $\text{m}^3$
  - const double `teaspoon` = 4.92892159375e-6  
 $\text{m}^3$
  - const double `canadian_gallon` = 4.54609e-3  
 $\text{m}^3$
-

- const double [uk\\_gallon](#) = 4.546092e-3  
 $m^3$
- const double [miles\\_per\\_hour](#) = 4.4704e-1  
 $m/s$
- const double [kilometers\\_per\\_hour](#) = 2.777777777778e-1  
 $m/s$
- const double [knot](#) = 5.144444444444e-1  
 $m/s$
- const double [pound\\_mass](#) = 4.5359237e-1  
 $kg$
- const double [ounce\\_mass](#) = 2.8349523125e-2  
 $kg$
- const double [ton](#) = 9.0718474e2  
 $kg$
- const double [metric\\_ton](#) = 1e3  
 $kg$
- const double [uk\\_ton](#) = 1.0160469088e3  
 $kg$
- const double [troy\\_ounce](#) = 3.1103475e-2  
 $kg$
- const double [carat](#) = 2e-4  
 $kg$
- const double [unified\\_atomic\\_mass](#) = 1.66053873e-27  
 $kg$
- const double [gram\\_force](#) = 9.80665e-3  
 $kg\ m/s^2$
- const double [pound\\_force](#) = 4.44822161526e0  
 $kg\ m/s^2$
- const double [kilopound\\_force](#) = 4.44822161526e3  
 $kg\ m/s^2$
- const double [poundal](#) = 1.38255e-1  
 $kg\ m/s^2$
- const double [calorie](#) = 4.1868e0  
 $kg\ m^2/s^2$
- const double [btu](#) = 1.05505585262e3  
 $kg\ m^2/s^2$
- const double [therm](#) = 1.05506e8  
 $kg\ m^2/s^2$
- const double [horsepower](#) = 7.457e2  
 $kg\ m^2/s^3$
- const double [bar](#) = 1e5  
 $kg/m\ s^2$
- const double [std\\_atmosphere](#) = 1.01325e5  
 $kg/m\ s^2$
- const double [torr](#) = 1.33322368421e2  
 $kg/m\ s^2$
- const double [meter\\_of\\_mercury](#) = 1.33322368421e5  
 $kg/m\ s^2$
- const double [inch\\_of\\_mercury](#) = 3.38638815789e3  
 $kg/m\ s^2$
- const double [inch\\_of\\_water](#) = 2.490889e2  
 $kg/m\ s^2$
- const double [psi](#) = 6.89475729317e3  
 $kg/m\ s^2$
- const double [poise](#) = 1e-1  
 $kg\ m^{-1}\ s^{-1}$
- const double [stokes](#) = 1e-4  
 $m^2/s$

- const double [faraday](#) = 9.6485341472e4  
*A s / mol.*
- const double [electron\\_charge](#) = 1.602176462e-19  
*A s.*
- const double [gauss](#) = 1e-4  
*kg / A s<sup>2</sup>*
- const double [stilb](#) = 1e4  
*cd / m<sup>2</sup>*
- const double [lumen](#) = 1e0  
*cd sr*
- const double [lux](#) = 1e0  
*cd sr / m<sup>2</sup>*
- const double [phot](#) = 1e4  
*cd sr / m<sup>2</sup>*
- const double [footcandle](#) = 1.076e1  
*cd sr / m<sup>2</sup>*
- const double [lambert](#) = 1e4  
*cd sr / m<sup>2</sup>*
- const double [footlambert](#) = 1.07639104e1  
*cd sr / m<sup>2</sup>*
- const double [curie](#) = 3.7e10  
*1 / s*
- const double [roentgen](#) = 2.58e-4  
*A s / kg.*
- const double [rad](#) = 1e-2  
*m<sup>2</sup> / s<sup>2</sup>*
- const double [solar\\_mass](#) = 1.98892e30  
*kg*
- const double [bohr\\_radius](#) = 5.291772083e-11  
*m*
- const double [newton](#) = 1e0  
*kg m / s<sup>2</sup>*
- const double [dyne](#) = 1e-5  
*kg m / s<sup>2</sup>*
- const double [joule](#) = 1e0  
*kg m<sup>2</sup> / s<sup>2</sup>*
- const double [erg](#) = 1e-7  
*kg m<sup>2</sup> / s<sup>2</sup>*
- const double [stefan\\_boltzmann\\_constant](#) = 5.67039934436e-8  
*kg / K<sup>4</sup> s<sup>3</sup>*
- const double [thomson\\_cross\\_section](#) = 6.65245853542e-29  
*m<sup>2</sup>*
- const double [vacuum\\_permittivity](#) = 8.854187817e-12  
*A<sup>2</sup> s<sup>4</sup> / kg m<sup>3</sup>.*
- const double [vacuum\\_permeability](#) = 1.25663706144e-6  
*kg m / A<sup>2</sup> s<sup>2</sup>*

## 2.4 gsl\_mkisa Namespace Reference

### 2.4.1 Detailed Description

GSL constants in MKSA units.

The MKSA units are given below each constant



## Variables

- const double [schwarzschild\\_radius](#) = 2.95325008e3  
*m*
- const double [speed\\_of\\_light](#) = 2.99792458e8  
*m / s*
- const double [gravitational\\_constant](#) = 6.673e-11  
*m<sup>3</sup> / kg s<sup>2</sup>*
- const double [plancks\\_constant\\_h](#) = 6.62606876e-34  
*kg m<sup>2</sup> / s*
- const double [plancks\\_constant\\_hbar](#) = 1.05457159642e-34  
*kg m<sup>2</sup> / s*
- const double [astronomical\\_unit](#) = 1.49597870691e11  
*m*
- const double [light\\_year](#) = 9.46053620707e15  
*m*
- const double [parsec](#) = 3.08567758135e16  
*m*
- const double [grav\\_accel](#) = 9.80665e0  
*m / s<sup>2</sup>*
- const double [electron\\_volt](#) = 1.602176462e-19  
*kg m<sup>2</sup> / s<sup>2</sup>*
- const double [mass\\_electron](#) = 9.10938188e-31  
*kg*
- const double [mass\\_muon](#) = 1.88353109e-28  
*kg*
- const double [mass\\_proton](#) = 1.67262158e-27  
*kg*
- const double [mass\\_neutron](#) = 1.67492716e-27  
*kg*
- const double [rydberg](#) = 2.17987190389e-18  
*kg m<sup>2</sup> / s<sup>2</sup>*
- const double [boltzmann](#) = 1.3806503e-23  
*kg m<sup>2</sup> / K s<sup>2</sup>*
- const double [bohr\\_magneton](#) = 9.27400899e-24  
*A m<sup>2</sup>.*
- const double [nuclear\\_magneton](#) = 5.05078317e-27  
*A m<sup>2</sup>.*
- const double [electron\\_magnetic\\_moment](#) = 9.28476362e-24  
*A m<sup>2</sup>.*
- const double [proton\\_magnetic\\_moment](#) = 1.410606633e-26  
*A m<sup>2</sup>.*
- const double [molar\\_gas](#) = 8.314472e0  
*kg m<sup>2</sup> / K mol s<sup>2</sup>*
- const double [standard\\_gas\\_volume](#) = 2.2710981e-2  
*m<sup>3</sup> / mol*
- const double [minute](#) = 6e1  
*s*
- const double [hour](#) = 3.6e3  
*s*
- const double [day](#) = 8.64e4  
*s*
- const double [week](#) = 6.048e5  
*s*
- const double [inch](#) = 2.54e-2  
*m*
- const double [foot](#) = 3.048e-1  
*m*

- const double [yard](#) = 9.144e-1  
*m*
- const double [mile](#) = 1.609344e3  
*m*
- const double [nautical\\_mile](#) = 1.852e3  
*m*
- const double [fathom](#) = 1.8288e0  
*m*
- const double [mil](#) = 2.54e-5  
*m*
- const double [point](#) = 3.52777777778e-4  
*m*
- const double [texpoint](#) = 3.51459803515e-4  
*m*
- const double [micron](#) = 1e-6  
*m*
- const double [angstrom](#) = 1e-10  
*m*
- const double [hectare](#) = 1e4  
*m*<sup>2</sup>
- const double [acre](#) = 4.04685642241e3  
*m*<sup>2</sup>
- const double [barn](#) = 1e-28  
*m*<sup>2</sup>
- const double [liter](#) = 1e-3  
*m*<sup>3</sup>
- const double [us\\_gallon](#) = 3.78541178402e-3  
*m*<sup>3</sup>
- const double [quart](#) = 9.46352946004e-4  
*m*<sup>3</sup>
- const double [pint](#) = 4.73176473002e-4  
*m*<sup>3</sup>
- const double [cup](#) = 2.36588236501e-4  
*m*<sup>3</sup>
- const double [fluid\\_ounce](#) = 2.95735295626e-5  
*m*<sup>3</sup>
- const double [tablespoon](#) = 1.47867647813e-5  
*m*<sup>3</sup>
- const double [teaspoon](#) = 4.92892159375e-6  
*m*<sup>3</sup>
- const double [canadian\\_gallon](#) = 4.54609e-3  
*m*<sup>3</sup>
- const double [uk\\_gallon](#) = 4.546092e-3  
*m*<sup>3</sup>
- const double [miles\\_per\\_hour](#) = 4.4704e-1  
*m* / *s*
- const double [kilometers\\_per\\_hour](#) = 2.77777777778e-1  
*m* / *s*
- const double [knot](#) = 5.14444444444e-1  
*m* / *s*
- const double [pound\\_mass](#) = 4.5359237e-1  
*kg*
- const double [ounce\\_mass](#) = 2.8349523125e-2  
*kg*
- const double [ton](#) = 9.0718474e2  
*kg*
- const double [metric\\_ton](#) = 1e3  
*kg*

- const double [uk\\_ton](#) = 1.0160469088e3  
*kg*
- const double [troy\\_ounce](#) = 3.1103475e-2  
*kg*
- const double [carat](#) = 2e-4  
*kg*
- const double [unified\\_atomic\\_mass](#) = 1.66053873e-27  
*kg*
- const double [gram\\_force](#) = 9.80665e-3  
*kg m / s^2*
- const double [pound\\_force](#) = 4.44822161526e0  
*kg m / s^2*
- const double [kilopound\\_force](#) = 4.44822161526e3  
*kg m / s^2*
- const double [poundal](#) = 1.38255e-1  
*kg m / s^2*
- const double [calorie](#) = 4.1868e0  
*kg m^2 / s^2*
- const double [btu](#) = 1.05505585262e3  
*kg m^2 / s^2*
- const double [therm](#) = 1.05506e8  
*kg m^2 / s^2*
- const double [horsepower](#) = 7.457e2  
*kg m^2 / s^3*
- const double [bar](#) = 1e5  
*kg / m s^2*
- const double [std\\_atmosphere](#) = 1.01325e5  
*kg / m s^2*
- const double [torr](#) = 1.33322368421e2  
*kg / m s^2*
- const double [meter\\_of\\_mercury](#) = 1.33322368421e5  
*kg / m s^2*
- const double [inch\\_of\\_mercury](#) = 3.38638815789e3  
*kg / m s^2*
- const double [inch\\_of\\_water](#) = 2.490889e2  
*kg / m s^2*
- const double [psi](#) = 6.89475729317e3  
*kg / m s^2*
- const double [poise](#) = 1e-1  
*kg m^-1 s^-1*
- const double [stokes](#) = 1e-4  
*m^2 / s*
- const double [faraday](#) = 9.6485341472e4  
*A s / mol.*
- const double [electron\\_charge](#) = 1.602176462e-19  
*A s.*
- const double [gauss](#) = 1e-4  
*kg / A s^2*
- const double [stilb](#) = 1e4  
*cd / m^2*
- const double [lumen](#) = 1e0  
*cd sr*
- const double [lux](#) = 1e0  
*cd sr / m^2*
- const double [phot](#) = 1e4  
*cd sr / m^2*
- const double [footcandle](#) = 1.076e1  
*cd sr / m^2*

- const double [lambert](#) = 1e4  
 $cd\ sr / m^2$
- const double [footlambert](#) = 1.07639104e1  
 $cd\ sr / m^2$
- const double [curie](#) = 3.7e10  
 $1 / s$
- const double [roentgen](#) = 2.58e-4  
 $A\ s / kg.$
- const double [rad](#) = 1e-2  
 $m^2 / s^2$
- const double [solar\\_mass](#) = 1.98892e30  
 $kg$
- const double [bohr\\_radius](#) = 5.291772083e-11  
 $m$
- const double [newton](#) = 1e0  
 $kg\ m / s^2$
- const double [dyne](#) = 1e-5  
 $kg\ m / s^2$
- const double [joule](#) = 1e0  
 $kg\ m^2 / s^2$
- const double [erg](#) = 1e-7  
 $kg\ m^2 / s^2$
- const double [stefan\\_boltzmann\\_constant](#) = 5.67039934436e-8  
 $kg / K^4\ s^3$
- const double [thomson\\_cross\\_section](#) = 6.65245853542e-29  
 $m^2$
- const double [vacuum\\_permittivity](#) = 8.854187817e-12  
 $A^2\ s^4 / kg\ m^3.$
- const double [vacuum\\_permeability](#) = 1.25663706144e-6  
 $kg\ m / A^2\ s^2$

## 2.5 gsl\_num Namespace Reference

### 2.5.1 Detailed Description

GSL numerical constants.

#### Variables

- const double [fine\\_structure](#) = 7.297352533e-3
- const double [avogadro](#) = 6.02214199e23
- const double [yotta](#) = 1e24
- const double [zetta](#) = 1e21
- const double [exa](#) = 1e18
- const double [peta](#) = 1e15
- const double [tera](#) = 1e12
- const double [giga](#) = 1e9
- const double [mega](#) = 1e6
- const double [kilo](#) = 1e3
- const double [milli](#) = 1e-3
- const double [micro](#) = 1e-6
- const double [nano](#) = 1e-9
- const double [pico](#) = 1e-12
- const double [femto](#) = 1e-15
- const double [atto](#) = 1e-18
- const double [zepto](#) = 1e-21
- const double [yocto](#) = 1e-24

## 2.6 o2scl\_arith Namespace Reference

### 2.6.1 Detailed Description

A namespace for arithmetic on complex numbers and vectors.

#### Functions

- template<class vec\_t, class vec2\_t>  
void [vector\\_copy](#) (size\_t N, vec\_t &v, vec2\_t &v2)  
*Naive vector copy.*
- template<class mat\_t, class mat2\_t>  
void [matrix\\_copy](#) (size\_t M, size\_t N, mat\_t &m, mat2\_t &m2)  
*Naive matrix copy.*
- template<class vec\_t, class vec2\_t>  
void [vector\\_cx\\_copy](#) (size\_t N, vec\_t &v, vec2\_t &v2)  
*Naive complex vector copy.*
- template<class mat\_t, class mat2\_t>  
void [matrix\\_cx\\_copy](#) (size\_t M, size\_t N, mat\_t &m, mat2\_t &m2)  
*Naive complex matrix copy.*

#### Binary operators for two complex numbers

- gsl\_complex [operator+](#) (gsl\_complex x, gsl\_complex y)  
*Add two complex numbers.*
- gsl\_complex [operator-](#) (gsl\_complex x, gsl\_complex y)  
*Subtract two complex numbers.*
- gsl\_complex [operator\\*](#) (gsl\_complex x, gsl\_complex y)  
*Multiply two complex numbers.*
- gsl\_complex [operator/](#) (gsl\_complex x, gsl\_complex y)  
*Divide two complex numbers.*

#### Binary operators with assignment for two complex numbers

- gsl\_complex [operator+=](#) (gsl\_complex &x, gsl\_complex y)  
*Add a complex number.*
- gsl\_complex [operator-=](#) (gsl\_complex &x, gsl\_complex y)  
*Subtract a complex number.*
- gsl\_complex [operator\\*=](#) (gsl\_complex &x, gsl\_complex y)  
*Multiply a complex number.*
- gsl\_complex [operator/=](#) (gsl\_complex &x, gsl\_complex y)  
*Divide a complex number.*

#### Binary operators with assignment for a complex and real

- gsl\_complex [operator+](#) (gsl\_complex x, double y)  
*Add a complex and real number.*
- gsl\_complex [operator+](#) (double y, gsl\_complex x)  
*Add a complex and real number.*
- gsl\_complex [operator-](#) (gsl\_complex x, double y)  
*Subtract a complex and real number.*
- gsl\_complex [operator-](#) (double y, gsl\_complex x)  
*Subtract a complex and real number.*
- gsl\_complex [operator\\*](#) (gsl\_complex x, double y)  
*Multiply a complex and real number.*
- gsl\_complex [operator\\*](#) (double y, gsl\_complex x)  
*Multiply a complex and real number.*
- gsl\_complex [operator/](#) (gsl\_complex x, double y)

*Divide a complex and real number.*

### Miscellaneous functions

- double **arg** (gsl\_complex x)
- double **abs** (gsl\_complex x)
- double **abs2** (gsl\_complex z)
- gsl\_complex **conjugate** (gsl\_complex a)

### Square root and exponent functions

- gsl\_complex **sqrt** (gsl\_complex a)
- gsl\_complex **sqrt\_real** (double x)
- gsl\_complex **pow** (gsl\_complex a, gsl\_complex b)
- gsl\_complex **pow\_real** (gsl\_complex a, double b)

### Logarithmic and exponential functions

- double **logabs** (gsl\_complex z)
- gsl\_complex **exp** (gsl\_complex a)
- gsl\_complex **log** (gsl\_complex a)
- gsl\_complex **log10** (gsl\_complex a)
- gsl\_complex **log\_b** (gsl\_complex a, gsl\_complex b)

### Trigonometric functions

- gsl\_complex **sin** (gsl\_complex a)
- gsl\_complex **cos** (gsl\_complex a)
- gsl\_complex **tan** (gsl\_complex a)
- gsl\_complex **sec** (gsl\_complex a)
- gsl\_complex **csc** (gsl\_complex a)
- gsl\_complex **cot** (gsl\_complex a)
- gsl\_complex **asin** (gsl\_complex a)
- gsl\_complex **asin\_real** (double a)
- gsl\_complex **acos** (gsl\_complex a)
- gsl\_complex **acos\_real** (double a)
- gsl\_complex **atan** (gsl\_complex a)
- gsl\_complex **asec** (gsl\_complex a)
- gsl\_complex **asec\_real** (double a)
- gsl\_complex **acsc** (gsl\_complex a)
- gsl\_complex **acsc\_real** (double a)
- gsl\_complex **acot** (gsl\_complex a)

### Hyperbolic trigonometric functions

- gsl\_complex **sinh** (gsl\_complex a)
- gsl\_complex **cosh** (gsl\_complex a)
- gsl\_complex **tanh** (gsl\_complex a)
- gsl\_complex **sech** (gsl\_complex a)
- gsl\_complex **csch** (gsl\_complex a)
- gsl\_complex **coth** (gsl\_complex a)
- gsl\_complex **asinh** (gsl\_complex a)
- gsl\_complex **acosh** (gsl\_complex a)
- gsl\_complex **acosh\_real** (double a)
- gsl\_complex **atanh** (gsl\_complex a)
- gsl\_complex **atanh\_real** (double a)
- gsl\_complex **asech** (gsl\_complex a)
- gsl\_complex **acsch** (gsl\_complex a)
- gsl\_complex **acoth** (gsl\_complex a)

## 2.7 o2scl\_const Namespace Reference

### 2.7.1 Detailed Description

O2scl constants.

## Variables

- const double `pi` =  $\text{acos}(-1.0)$   
 $\pi$
- const double `pi2` = `pi*pi`  
 $\pi^2$
- const double `zeta32` = 2.6123753486854883433  
 $\zeta(3/2)$
- const double `zeta2` = 1.6449340668482264365  
 $\zeta(2)$
- const double `zeta52` = 1.3414872572509171798  
 $\zeta(5/2)$
- const double `zeta3` = 1.2020569031595942854  
 $\zeta(3)$
- const double `zeta5` = 1.0369277551433699263  
 $\zeta(5)$
- const double `zeta7` = 1.0083492773819228268  
 $\zeta(7)$

## Particle Physics Booklet

(see also D.E. Groom, et. al., Euro. Phys. J. C 15 (2000) 1.)

- const double `hc_mev_fm` = 197.3269602  
 $\hbar c$  in MeV fm
- const double `sin2_theta_weak` = 0.2224  
 $\sin^2 \theta_W$
- const double `gfermi_gev` = 1.16639e-5  
Fermi coupling constant ( $G_F$ ) in  $\text{GeV}^{-2}$ .
- const double `mev_kg` = 1.782661731e-30  
1 MeV in kg
- const double `ev_mks` = 1.602176462e-19  
1 eV in  $\text{kg} \cdot \text{m}^2/\text{s}^2$  (Joules)
- const double `mev_cgs` = 1.60217733e-6  
1 MeV in  $\text{g} \cdot \text{cm}^2/\text{s}^2$  (ergs)
- const double `hc_mev_cm` = 1.973269602e-11  
 $\hbar c$  in MeV cm
- const double `boltzmann_mev_K` = 8.617342e-11  
1 MeV in Kelvin

## Squared electron charge

- const double `e2_gaussian` = `o2scl_const::hc_mev_fm*gsl_num::fine_structure`  
Electron charge squared in Gaussian units.
- const double `e2_hlorentz` = `gsl_num::fine_structure*4.0*pi`  
Electron charge squared in Heaviside-Lorentz units where  $\hbar = c = 1$ .
- const double `e2_mkpa` = `gsl_mkpa::electron_charge`  
Electron charge squared in SI(MKSA) units.

### 2.7.2 Variable Documentation

#### 2.7.2.1 const double e2\_gaussian = o2scl\_const::hc\_mev\_fm\*gsl\_num::fine\_structure

Electron charge squared in Gaussian units.

In Gaussian Units:

$$\vec{\nabla} \cdot \vec{E} = 4\pi\rho, \quad \vec{E} = -\vec{\nabla}\Phi, \quad \nabla^2\Phi = -4\pi\rho,$$

$$F = \frac{q_1 q_2}{r^2}, \quad W = \frac{1}{2} \int \rho V d^3x = \frac{1}{8\pi} \int |\vec{E}|^2 d^3x, \quad \alpha = \frac{e^2}{\hbar c} = \frac{1}{137}$$

Definition at line 955 of file constants.h.

**2.7.2.2 const double e2\_hlorentz = gsl\_num::fine\_structure\*4.0\*pi**

Electron charge squared in Heaviside-Lorentz units where  $\hbar = c = 1$ .

In Heaviside-Lorentz units:

$$\vec{\nabla} \cdot \vec{E} = \rho, \quad \vec{E} = -\vec{\nabla}\Phi, \quad \nabla^2\Phi = -\rho, \\ F = \frac{q_1 q_2}{4\pi r^2}, \quad W = \frac{1}{2} \int \rho V d^3x = \frac{1}{2} \int |\vec{E}|^2 d^3x, \quad \alpha = \frac{e^2}{4\pi} = \frac{1}{137}$$

Definition at line 975 of file constants.h.

**2.7.2.3 const double e2\_mkسا = gsl\_mkسا::electron\_charge**

Electron charge squared in SI(MKSA) units.

In MKSA units:

$$\vec{\nabla} \cdot \vec{E} = \rho, \quad \vec{E} = -\vec{\nabla}\Phi, \quad \nabla^2\Phi = -\rho, \\ F = \frac{1}{4\pi\epsilon_0} \frac{q_1 q_2}{r^2}, \quad W = \frac{1}{2} \int \rho V d^3x = \frac{\epsilon_0}{2} \int |\vec{E}|^2 d^3x, \quad \alpha = \frac{e^2}{4\pi\epsilon_0\hbar c} = \frac{1}{137}$$

Note the conversion formulas

$$q_H L = \sqrt{4\pi} q_G = \frac{1}{\sqrt{\epsilon_0}} q_{SI}$$

as mentioned in pg. 13 of D. Griffiths Intro to Elem. Particles.

Definition at line 1001 of file constants.h.

**2.8 o2scl\_fm Namespace Reference****2.8.1 Detailed Description**

Constants in units of fm.

In nuclear physics is frequently convenient to work in units of fm with  $\hbar = c = k_B = 1$ . Several useful constants are given here.

For example, `mev` gives 1 MeV in units of  $\text{fm}^{-1}$  (the solution to the equation  $1\text{MeV} = x \text{fm}^{-1}$ ). If you have a number in MeV, you can multiply by `mev` to get a number in units of  $\text{fm}^{-1}$ . Alternatively, `mev` is a number with units  $\text{MeV}^{-1} \cdot \text{fm}^{-1}$ . These can be combined, so that `erg` divided by `sec` is 1 erg/sec in units of  $\text{fm}^{-2}$ .

**Variables**

- const double `mev` = 1.0/o2scl\_const::hc\_mev\_fm  
1 MeV in  $\text{fm}^{-1}$
- const double `kg` = mev/1.782661731e-30  
1 kg in  $\text{fm}^{-1}$
- const double `msun_per_km3` = gsl\_mks::solar\_mass/1.0e54\*kg  
1  $M_\odot/\text{km}^3$  in  $\text{fm}^{-4}$
- const double `Kelvin` = 8.617342e-11\*mev  
1 Kelvin in  $\text{fm}^{-1}$
- const double `joule` = kg/gsl\_mks::speed\_of\_light/gsl\_mks::speed\_of\_light  
1 Joule in  $\text{fm}^{-1}$
- const double `erg` = kg/1.0e3/gsl\_cgs::speed\_of\_light/gsl\_cgs::speed\_of\_light  
1 erg in  $\text{fm}^{-1}$
- const double `sec` = gsl\_mks::speed\_of\_light\*1.0e15  
1 second in fm



### Masses from Particle Physics Booklet

(see also D.E. Groom, et. al., Euro. Phys. J. C 15 (2000) 1.)

- const double `mass_electron` = 0.510998902/o2scl\_const::hc\_mev\_fm  
*Electron mass in fm<sup>-1</sup>.*
- const double `mass_muon` = 105.658357/o2scl\_const::hc\_mev\_fm  
*Muon mass in fm<sup>-1</sup>.*
- const double `mass_amu` = 931.494013/o2scl\_const::hc\_mev\_fm  
*Atomic mass unit in fm<sup>-1</sup>.*
- const double `mass_neutron` = 939.565/o2scl\_const::hc\_mev\_fm  
*Neutron mass in fm<sup>-1</sup>.*
- const double `mass_proton` = 938.272/o2scl\_const::hc\_mev\_fm  
*Proton mass in fm<sup>-1</sup>.*
- const double `mass_lambda` = 1115.683/o2scl\_const::hc\_mev\_fm  
*Λ mass in fm<sup>-1</sup>.*
- const double `mass_sigmam` = 1197.45/o2scl\_const::hc\_mev\_fm  
*Σ<sup>-</sup> mass in fm<sup>-1</sup>.*
- const double `mass_sigma` = 1192.642/o2scl\_const::hc\_mev\_fm  
*Σ<sup>0</sup> mass in fm<sup>-1</sup>.*
- const double `mass_sigmap` = 1189.37/o2scl\_const::hc\_mev\_fm  
*Σ<sup>+</sup> mass in fm<sup>-1</sup>.*
- const double `mass_cascadem` = 1321.3/o2scl\_const::hc\_mev\_fm  
*Ξ<sup>-</sup> mass in fm<sup>-1</sup>.*
- const double `mass_cascade` = 1314.8/o2scl\_const::hc\_mev\_fm  
*Ξ<sup>0</sup> mass in fm<sup>-1</sup>.*
- const double `mass_omega` = 782.57/o2scl\_const::hc\_mev\_fm  
*ω mass in fm<sup>-1</sup>.*
- const double `mass_rho` = 769.3/o2scl\_const::hc\_mev\_fm  
*ρ mass in fm<sup>-1</sup>.*

#### www.nist.gov

- const double `mass_alpha` = 3727.37905/o2scl\_const::hc\_mev\_fm  
*Alpha particle mass in fm<sup>-1</sup>.*

## 2.8.2 Variable Documentation

### 2.8.2.1 const double mass\_alpha = 3727.37905/o2scl\_const::hc\_mev\_fm

Alpha particle mass in fm<sup>-1</sup>.

This does not include the mass of the additional two electrons which are present in a helium atom.

Definition at line 1066 of file constants.h.

## 2.9 o2scl\_inte\_qag\_coeffs Namespace Reference

### 2.9.1 Detailed Description

A namespace for the GSL adaptive integration coefficients.

#### Documentation from GSL:

Gauss quadrature weights and kronrod quadrature abscissae and weights as evaluated with 80 decimal digit arithmetic by L. W. Fullerton, Bell Labs, Nov. 1981.

#### Variables

- static const double `qk15_xgk` [8]

- static const double [qk15\\_wg](#) [4]
- static const double [qk15\\_wgk](#) [8]
- static const double [qk21\\_xgk](#) [11]
- static const double [qk21\\_wg](#) [5]
- static const double [qk21\\_wgk](#) [11]
- static const double [qk31\\_xgk](#) [16]
- static const double [qk31\\_wg](#) [8]
- static const double [qk31\\_wgk](#) [16]
- static const double [qk41\\_xgk](#) [21]
- static const double [qk41\\_wg](#) [11]
- static const double [qk41\\_wgk](#) [21]
- static const double [qk51\\_xgk](#) [26]
- static const double [qk51\\_wg](#) [13]
- static const double [qk51\\_wgk](#) [26]
- static const double [qk61\\_xgk](#) [31]
- static const double [qk61\\_wg](#) [15]
- static const double [qk61\\_wgk](#) [31]

## 2.9.2 Variable Documentation

### 2.9.2.1 const double [qk15\\_wg](#)[4] [static]

weights of the 7-point gauss rule

Definition at line 59 of file `gsl_inte_qag_b.h`.

### 2.9.2.2 const double [qk15\\_wgk](#)[8] [static]

weights of the 15-point kronrod rule

Definition at line 68 of file `gsl_inte_qag_b.h`.

### 2.9.2.3 const double [qk15\\_xgk](#)[8] [static]

abscissae of the 15-point kronrod rule

Definition at line 42 of file `gsl_inte_qag_b.h`.

### 2.9.2.4 const double [qk21\\_wg](#)[5] [static]

weights of the 10-point gauss rule

Definition at line 101 of file `gsl_inte_qag_b.h`.

### 2.9.2.5 const double [qk21\\_wgk](#)[11] [static]

weights of the 21-point kronrod rule

Definition at line 111 of file `gsl_inte_qag_b.h`.

### 2.9.2.6 const double [qk21\\_xgk](#)[11] [static]

abscissae of the 21-point kronrod rule

Definition at line 81 of file `gsl_inte_qag_b.h`.

### 2.9.2.7 const double [qk31\\_wg](#)[8] [static]

weights of the 15-point gauss rule

Definition at line 152 of file `gsl_inte_qag_b.h`.

---

**2.9.2.8 const double qk31\_wgk[16] [static]**

weights of the 31-point kronrod rule

Definition at line 165 of file gsl\_inte\_qag\_b.h.

**2.9.2.9 const double qk31\_xgk[16] [static]**

abscissae of the 31-point kronrod rule

Definition at line 127 of file gsl\_inte\_qag\_b.h.

**2.9.2.10 const double qk41\_wg[11] [static]**

weights of the 20-point gauss rule

Definition at line 216 of file gsl\_inte\_qag\_b.h.

**2.9.2.11 const double qk41\_wgk[21] [static]**

weights of the 41-point kronrod rule

Definition at line 231 of file gsl\_inte\_qag\_b.h.

**2.9.2.12 const double qk41\_xgk[21] [static]**

abscissae of the 41-point kronrod rule

Definition at line 186 of file gsl\_inte\_qag\_b.h.

**2.9.2.13 const double qk51\_wg[13] [static]**

weights of the 25-point gauss rule

Definition at line 292 of file gsl\_inte\_qag\_b.h.

**2.9.2.14 const double qk51\_wgk[26] [static]**

weights of the 51-point kronrod rule

Definition at line 310 of file gsl\_inte\_qag\_b.h.

**2.9.2.15 const double qk51\_xgk[26] [static]**

abscissae of the 51-point kronrod rule

Definition at line 257 of file gsl\_inte\_qag\_b.h.

**2.9.2.16 const double qk61\_wg[15] [static]**

weights of the 30-point gauss rule

Definition at line 383 of file gsl\_inte\_qag\_b.h.

**2.9.2.17 const double qk61\_wgk[31] [static]**

weights of the 61-point kronrod rule

Definition at line 403 of file gsl\_inte\_qag\_b.h.

---

### 2.9.2.18 const double qk61\_xgk[31] [static]

abscissae of the 61-point kronrod rule

Definition at line 343 of file gsl\_inte\_qag\_b.h.

## 2.10 o2scl\_inte\_qng\_coeffs Namespace Reference

### 2.10.1 Detailed Description

A namespace for the quadrature coefficients for non-adaptive integration.

#### Documentation from GSL:

Gauss-Kronrod-Patterson quadrature coefficients for use in quadpack routine qng. These coefficients were calculated with 101 decimal digit arithmetic by L. W. Fullerton, Bell Labs, Nov 1981.

#### Variables

- static const double [x1](#) [5]
- static const double [w10](#) [5]
- static const double [x2](#) [5]
- static const double [w21a](#) [5]
- static const double [w21b](#) [6]
- static const double [x3](#) [11]
- static const double [w43a](#) [10]
- static const double [w43b](#) [12]
- static const double [x4](#) [22]
- static const double [w87a](#) [21]
- static const double [w87b](#) [23]

### 2.10.2 Variable Documentation

#### 2.10.2.1 const double w10[5] [static]

w10, weights of the 10-point formula

Definition at line 51 of file gsl\_inte\_qng.h.

#### 2.10.2.2 const double w21a[5] [static]

w21a, weights of the 21-point formula for abscissae x1

Definition at line 69 of file gsl\_inte\_qng.h.

#### 2.10.2.3 const double w21b[6] [static]

w21b, weights of the 21-point formula for abscissae x2

Definition at line 78 of file gsl\_inte\_qng.h.

#### 2.10.2.4 const double w43a[10] [static]

w43a, weights of the 43-point formula for abscissae x1, x3

Definition at line 103 of file gsl\_inte\_qng.h.

---

**2.10.2.5 const double w43b[12] [static]**

w43b, weights of the 43-point formula for abscissae x3

Definition at line 117 of file gsl\_inte\_qng.h.

**2.10.2.6 const double w87a[21] [static]**

w87a, weights of the 87-point formula for abscissae x1, x2, x3

Definition at line 159 of file gsl\_inte\_qng.h.

**2.10.2.7 const double w87b[23] [static]**

w87b, weights of the 87-point formula for abscissae x4

Definition at line 184 of file gsl\_inte\_qng.h.

**2.10.2.8 const double x1[5] [static]**

x1, abscissae common to the 10-, 21-, 43- and 87-point rule

Definition at line 42 of file gsl\_inte\_qng.h.

**2.10.2.9 const double x2[5] [static]**

x2, abscissae common to the 21-, 43- and 87-point rule

Definition at line 60 of file gsl\_inte\_qng.h.

**2.10.2.10 const double x3[11] [static]**

x3, abscissae common to the 43- and 87-point rule

Definition at line 88 of file gsl\_inte\_qng.h.

**2.10.2.11 const double x4[22] [static]**

x4, abscissae of the 87-point rule

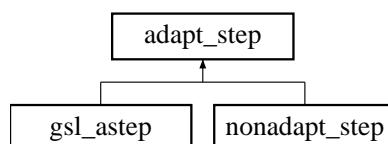
Definition at line 133 of file gsl\_inte\_qng.h.

## 3 O2scl Data Structure Documentation

### 3.1 adapt\_step Class Template Reference

```
#include <adapt_step.h>
```

Inheritance diagram for adapt\_step::



### 3.1.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>
class adapt_step< param_t, func_t, vec_t, alloc_vec_t, alloc_t >
```

Adaptive stepper base.

The adaptive stepper is based on [gsl\\_rkck](#) unless otherwise specified in [set\\_step\(\)](#).

Definition at line 42 of file [adapt\\_step.h](#).

#### Public Member Functions

- [adapt\\_step](#) ()
- virtual [~adapt\\_step](#) ()
- virtual int [astep](#) (double &x, double &h, double xmax, size\_t n, vec\_t &y, param\_t &pa, func\_t &derivs)  
*Make an adaptive integration step of the system `derivs`.*
- virtual int [astep\\_derivs](#) (double &x, double &h, double xmax, size\_t n, vec\_t &y, vec\_t &dydx, param\_t &pa, func\_t &derivs)  
*Make an adaptive integration step of the system `derivs` with derivatives.*
- int [set\\_step](#) ([odestep](#)< param\_t, func\_t, vec\_t > &step)  
*Set stepper.*

#### Data Fields

- int [verbose](#)  
*Set output level.*
- [gsl\\_rkck](#)< param\_t, func\_t, vec\_t, alloc\_vec\_t, alloc\_t > [def\\_step](#)  
*The default stepper.*

#### Protected Attributes

- [odestep](#)< param\_t, func\_t, vec\_t > \* [stepp](#)  
*Pointer to the stepper being used.*

### 3.1.2 Member Function Documentation

**3.1.2.1 virtual int astep (double & x, double & h, double xmax, size\_t n, vec\_t & y, param\_t & pa, func\_t & derivs)**  
[inline, virtual]

Make an adaptive integration step of the system `derivs`.

This attempts to take a step of size `h` from the point `x` of an `n`-dimensional system `derivs` starting with `y`. On exit, `x` and `y` contain the new values at the end of the step, `h` contains the size of the step.

Reimplemented in [gsl\\_astep](#), and [nonadapt\\_step](#).

Definition at line 62 of file [adapt\\_step.h](#).

**3.1.2.2 virtual int astep\_derivs (double & x, double & h, double xmax, size\_t n, vec\_t & y, vec\_t & dydx, param\_t & pa, func\_t & derivs)** [inline, virtual]

Make an adaptive integration step of the system `derivs` with derivatives.

This attempts to take a step of size `h` from the point `x` of an `n`-dimensional system `derivs` starting with `y` and given the initial derivatives `dydx`. On exit, `x`, `y` and `dydx` contain the new values at the end of the step, `h` contains the size of the step.

Reimplemented in [gsl\\_astep](#), and [nonadapt\\_step](#).

Definition at line 80 of file adapt\_step.h.

### 3.1.2.3 int set\_step (odestep< param\_t, func\_t, vec\_t > & step) [inline]

Set stepper.

This sets the stepper for use in the adaptive step routine. If no stepper is specified, then the default ([gsl\\_rkck](#)) is used.

Definition at line 99 of file adapt\_step.h.

The documentation for this class was generated from the following file:

- adapt\_step.h

## 3.2 array\_2d\_alloc Class Template Reference

```
#include <array.h>
```

### 3.2.1 Detailed Description

```
template<class vec_t> class array_2d_alloc< vec_t >
```

A simple class to provide an [allocate\(\)](#) function for 2-dimensional arrays.

The functions here are blank, as fixed-length arrays are automatically allocated and destroyed by the compiler. This class is present to provide a consistent analog to [pointer\\_alloc](#) and [ovector\\_alloc](#)

Definition at line 92 of file array.h.

#### Public Member Functions

- void [allocate](#) (vec\_t &v, int i, int j)  
*Allocate v for i elements.*
- void [free](#) (vec\_t &v)  
*Free memory.*

The documentation for this class was generated from the following file:

- [array.h](#)

## 3.3 array\_alloc Class Template Reference

```
#include <array.h>
```

### 3.3.1 Detailed Description

```
template<class vec_t> class array_alloc< vec_t >
```

A simple class to provide an [allocate\(\)](#) function for arrays.

The functions here are blank, as fixed-length arrays are automatically allocated and destroyed by the compiler. This class is present to provide a consistent analog to [pointer\\_alloc](#) and [ovector\\_alloc](#)

Definition at line 74 of file array.h.

### Public Member Functions

- void [allocate](#) (vec\_t &v, int i)  
*Allocate  $v$  for  $i$  elements.*
- void [free](#) (vec\_t &v)  
*Free memory.*

The documentation for this class was generated from the following file:

- [array.h](#)

## 3.4 array\_const\_reverse Class Template Reference

```
#include <array.h>
```

### 3.4.1 Detailed Description

**template<size\_t sz> class array\_const\_reverse< sz >**

A simple class which reverses the order of an array.

Definition at line 159 of file array.h.

### Public Member Functions

- [array\\_const\\_reverse](#) (const double \*arr)  
*Create a reversed array from arr of size sz.*
- const double & [operator](#)[ ] (size\_t i) const  
*Array-like indexing.*

### Protected Attributes

- double \* [a](#)  
*The array pointer.*

The documentation for this class was generated from the following file:

- [array.h](#)

## 3.5 array\_const\_subvector Class Reference

```
#include <array.h>
```

### 3.5.1 Detailed Description

A simple subvector class for a const array (without error checking).

Definition at line 238 of file array.h.

---



## Public Member Functions

- `array_const_subvector` (const double \*arr, size\_t offset, size\_t n)  
*Create a reversed array from `arr` of size `sz`.*
- const double & `operator[]` (size\_t i) const  
*Array-like indexing.*

## Protected Attributes

- double \* `a`  
*The array pointer.*
- size\_t `off`  
*The offset.*
- size\_t `len`  
*The subvector length.*

The documentation for this class was generated from the following file:

- `array.h`

## 3.6 `array_const_subvector_reverse` Class Reference

```
#include <array.h>
```

### 3.6.1 Detailed Description

Reverse a subvector of a const array.

Definition at line 325 of file `array.h`.

## Public Member Functions

- `array_const_subvector_reverse` (const double \*arr, size\_t offset, size\_t n)  
*Create a reversed array from `arr` of size `sz`.*
- const double & `operator[]` (size\_t i) const  
*Array-like indexing.*

## Protected Attributes

- double \* `a`  
*The array pointer.*
- size\_t `off`  
*The offset.*
- size\_t `len`  
*The subvector length.*

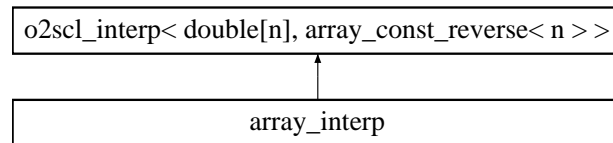
The documentation for this class was generated from the following file:

- `array.h`
-

## 3.7 array\_interp Class Template Reference

```
#include <interp.h>
```

Inheritance diagram for array\_interp::



### 3.7.1 Detailed Description

```
template<size_t n> class array_interp< n >
```

A specialization of [o2scl\\_interp](#) for C-style double arrays.

Definition at line 1091 of file interp.h.

#### Public Member Functions

- [array\\_interp](#) ([base\\_interp](#)< double[n]> &it, [base\\_interp](#)< [array\\_const\\_reverse](#)< n > > &rit)  
*Create with base interpolation objects it and rit.*
- [array\\_interp](#) ([base\\_interp](#)< double[n]> &it)  
*Create with base interpolation object it and use the default base interpolation object for reversed arrays.*
- [array\\_interp](#) ()  
*Create an interpolator using the default base interpolation objects.*

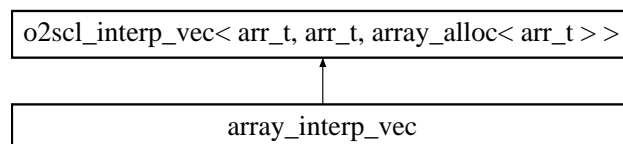
The documentation for this class was generated from the following file:

- interp.h

## 3.8 array\_interp\_vec Class Template Reference

```
#include <interp.h>
```

Inheritance diagram for array\_interp\_vec::



### 3.8.1 Detailed Description

```
template<class arr_t> class array_interp_vec< arr_t >
```

A specialization of [o2scl\\_interp\\_vec](#) for C-style arrays.

Definition at line 1121 of file interp.h.

## Public Member Functions

- [array\\_interp\\_vec](#) ([base\\_interp](#)< arr\_t > &it, size\_t nv, const arr\_t &x, const arr\_t &y)  
*Create with base interpolation object it.*

The documentation for this class was generated from the following file:

- [interp.h](#)

## 3.9 array\_reverse Class Template Reference

```
#include <array.h>
```

### 3.9.1 Detailed Description

```
template<size_t sz> class array_reverse< sz >
```

A simple class which reverses the order of an array.

Definition at line 120 of file array.h.

## Public Member Functions

- [array\\_reverse](#) (double \*arr)  
*Create a reversed array from arr of size sz.*
- double & [operator](#)[ ] (size\_t i)  
*Array-like indexing.*
- const double & [operator](#)[ ] (size\_t i) const  
*Array-like indexing.*

## Protected Attributes

- double \* [a](#)  
*The array pointer.*

The documentation for this class was generated from the following file:

- [array.h](#)

## 3.10 array\_row Class Template Reference

```
#include <array.h>
```

### 3.10.1 Detailed Description

```
template<class data_t, class array_2d_t> class array_row< data_t, array_2d_t >
```

Extract a row of a C-style 2d-array.

Definition at line 361 of file array.h.

## Public Member Functions

- [array\\_row](#) (array\_2d\_t &a, size\_t i)
- data\_t & [operator\[\]](#) (size\_t i)

## Data Fields

- data\_t \* [p](#)

The documentation for this class was generated from the following file:

- [array.h](#)

## 3.11 array\_subvector Class Reference

```
#include <array.h>
```

### 3.11.1 Detailed Description

A simple subvector class for an array (without error checking).

Definition at line 191 of file array.h.

## Public Member Functions

- [array\\_subvector](#) (double \*arr, size\_t offset, size\_t n)  
*Create a reversed array from arr of size sz.*
- double & [operator\[\]](#) (size\_t i)  
*Array-like indexing.*
- const double & [operator\[\]](#) (size\_t i) const  
*Array-like indexing.*

## Protected Attributes

- double \* [a](#)  
*The array pointer.*
- size\_t [off](#)  
*The offset.*
- size\_t [len](#)  
*The subvector length.*

The documentation for this class was generated from the following file:

- [array.h](#)

## 3.12 array\_subvector\_reverse Class Reference

```
#include <array.h>
```

---

### 3.12.1 Detailed Description

Reverse a subvector of an array.

Definition at line 278 of file array.h.

#### Public Member Functions

- [array\\_subvector\\_reverse](#) (double \*arr, size\_t offset, size\_t n)  
*Create a reversed array from arr of size sz.*
- double & [operator\[\]](#) (size\_t i)  
*Array-like indexing.*
- const double & [operator\[\]](#) (size\_t i) const  
*Array-like indexing.*

#### Protected Attributes

- double \* [a](#)  
*The array pointer.*
- size\_t [off](#)  
*The offset.*
- size\_t [len](#)  
*The subvector length.*

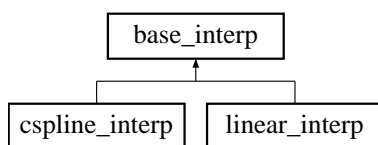
The documentation for this class was generated from the following file:

- [array.h](#)

## 3.13 base\_interp Class Template Reference

```
#include <interp.h>
```

Inheritance diagram for base\_interp::



### 3.13.1 Detailed Description

```
template<class vec_t> class base_interp< vec_t >
```

Base low-level interpolation class.

The descendants of this class are intended to be fast interpolation routines for increasing functions, leaving the some error handling, user-friendliness, and other more complicated improvements for other classes.

For any pair of vectors x and y into which you would like to interpolate, you need to call [alloc\(\)](#) and [init\(\)](#) first, and then the interpolation functions, and then [free\(\)](#). If the next pair of vectors has the same size, then you need only to call [init\(\)](#) before the next call to an interpolation function. If the vectors do not change, then you may call the interpolation functions in succession.

All of the descendants are based on the GSL interpolation routines and give identical results.

### Idea for future

These might work for decreasing functions by just replacing calls to `search_vec::bsearch_inc()` with `search_vec::bsearch_dec()`. If this is the case, then this should be rewritten accordingly.

Definition at line 65 of file `interp.h`.

### Public Member Functions

- virtual `~base_interp()`
- virtual `int alloc(size_t size)`  
*Allocate memory, assuming  $x$  and  $y$  have size `size`.*
- virtual `int free()`  
*Free allocated memory.*
- virtual `int init(const vec_t &x, const vec_t &y, size_t size)`  
*Initialize interpolation routine.*
- virtual `int interp(const vec_t &x, const vec_t &y, size_t size, double x0, double &y0)`  
*Give the value of the function  $y(x = x_0)$ .*
- virtual `int deriv(const vec_t &x, const vec_t &y, size_t size, double x0, double &dydx)`  
*Give the value of the derivative  $y'(x = x_0)$ .*
- virtual `int deriv2(const vec_t &x, const vec_t &y, size_t size, double x0, double &d2ydx2)`  
*Give the value of the second derivative  $y''(x = x_0)$ .*
- virtual `int integ(const vec_t &x, const vec_t &y, size_t size, double a, double b, double &result)`  
*Give the value of the integral  $\int_a^b y(x) dx$ .*

### Data Fields

- `size_t min_size`  
*The minimum size of the vectors to interpolate between.*

### Protected Attributes

- `search_vec<vec_t> sv`  
*The binary search object.*

## 3.13.2 Field Documentation

### 3.13.2.1 size\_t min\_size

The minimum size of the vectors to interpolate between.

This needs to be set in the constructor of the children for access by the class user

Definition at line 86 of file `interp.h`.

The documentation for this class was generated from the following file:

- `interp.h`

## 3.14 base\_ioc Class Reference

```
#include <base_ioc.h>
```

### 3.14.1 Detailed Description

Setup I/O objects for base library classes.

Definition at line 41 of file base\_ioc.h.

#### Public Member Functions

- [base\\_ioc \(\)](#)
- [~base\\_ioc \(\)](#)

#### Data Fields

- [bool\\_io\\_type \\* bool\\_io](#)
- [char\\_io\\_type \\* char\\_io](#)
- [double\\_io\\_type \\* double\\_io](#)
- [int\\_io\\_type \\* int\\_io](#)
- [long\\_io\\_type \\* long\\_io](#)
- [string\\_io\\_type \\* string\\_io](#)
- [word\\_io\\_type \\* word\\_io](#)
- [table\\_io\\_type \\* table\\_io](#)

The documentation for this class was generated from the following file:

- [base\\_ioc.h](#)

## 3.15 bin\_size Class Reference

```
#include <bin_size.h>
```

### 3.15.1 Detailed Description

Determine bin size (CERNLIB).

This is adapted from the KERNLIB routine `binsiz.f` written by F. James.

This class computes an appropriate set of histogram bins given the upper and lower limits of the data and the maximum number of bins. The bin width is always an integral power of ten times 1, 2, 2.5 or 5. The bin width may also be specified by the user, in which case the class only computes the appropriate limits.

#### Todo

Doesn't seem to be working yet.

Definition at line 47 of file bin\_size.h.

#### Public Member Functions

- [bin\\_size \(\)](#)
- [int calc\\_bin \(double al, double ah, int na, double &bl, double &bh, int &nb, double &bwid\)](#)  
*Compute bin size.*

## Data Fields

- bool `cern_mode`  
(default true)

### 3.15.2 Member Function Documentation

#### 3.15.2.1 int calc\_bin (double al, double ah, int na, double & bl, double & bh, int & nb, double & bwid)

Compute bin size.

- al - Lower limit of data
- ah - Upper limit of data
- na - Maximum number of bins desired.
- bl - Lower limit (BL<=AL)
- bh - Upper limit (BH>=AH)
- nb - Number of bins determined by BINSIZ ( $NA/2 < NB \leq NA$ )
- bwid - Bin width (BH-BL)/NB

If na=0 or na=-1, this function always makes exactly one bin.

If na=1, this function takes bwid as input and determines only bl, hb, and nb. This is especially useful when it is desired to have the same bin width for several histograms (or for the two axes of a scatter-plot).

If al > ah, this function takes al to be the upper limit and ah to be the lower limit, so that in fact al and ah may appear in any order. They are not changed by `calc_bin()`. If al = ah, the lower limit is taken to be al, and the upper limit is set to al+1.

If `cern_mode` is true (which is the default) the starting guess for the number of bins is na-1. Otherwise, the starting guess for the number of bins is na.

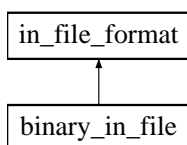
The documentation for this class was generated from the following file:

- bin\_size.h

## 3.16 binary\_in\_file Class Reference

```
#include <binary_file.h>
```

Inheritance diagram for binary\_in\_file::



### 3.16.1 Detailed Description

Binary input file.

Definition at line 137 of file binary\_file.h.



## Public Member Functions

- `binary_in_file` (std::string file\_name)  
*Read an input file with name file\_name.*
- virtual int `bool_in` (bool &dat, std::string name="")  
*Input a bool variable.*
- virtual int `char_in` (char &dat, std::string name="")  
*Input a char variable.*
- virtual int `double_in` (double &dat, std::string name="")  
*Input a double variable.*
- virtual int `float_in` (float &dat, std::string name="")  
*Input a float variable.*
- virtual int `int_in` (int &dat, std::string name="")  
*Input an int variable.*
- virtual int `long_in` (unsigned long int &dat, std::string name="")  
*Input an long variable.*
- virtual int `string_in` (std::string &dat, std::string name="")  
*Input a string variable.*
- virtual int `word_in` (std::string &dat, std::string name="")  
*Input a word variable.*
- virtual int `init_file` ()  
*Read the initialization.*
- virtual int `clean_up` ()  
*Clean up the file.*
- virtual int `start_object` (std::string &type, std::string &name)  
*Begin reading an object.*
- virtual int `skip_object` ()  
*Skip the present object for the next call to read\_type().*
- virtual int `end_object` ()  
*Finish reading an object.*

## Protected Attributes

- std::ifstream `ins`  
*The input stream.*

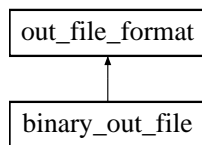
The documentation for this class was generated from the following file:

- `binary_file.h`

## 3.17 binary\_out\_file Class Reference

```
#include <binary_file.h>
```

Inheritance diagram for `binary_out_file`:



### 3.17.1 Detailed Description

Binary output file.

Definition at line 41 of file binary\_file.h.

#### Public Member Functions

- [binary\\_out\\_file](#) (std::string file\_name)  
*Create a binary output file with name file\_name.*
- virtual int [bool\\_out](#) (bool dat, std::string name="")  
*Output a bool variable.*
- virtual int [char\\_out](#) (char dat, std::string name="")  
*Output a char variable.*
- virtual int [double\\_out](#) (double dat, std::string name="")  
*Output a double variable.*
- virtual int [float\\_out](#) (float dat, std::string name="")  
*Output a float variable.*
- virtual int [int\\_out](#) (int dat, std::string name="")  
*Output an int variable.*
- virtual int [long\\_out](#) (unsigned long int dat, std::string name="")  
*Output an long variable.*
- virtual int [string\\_out](#) (std::string dat, std::string name="")  
*Output a string.*
- virtual int [word\\_out](#) (std::string dat, std::string name="")  
*Output a word.*
- virtual int [start\\_object](#) (std::string type, std::string name="")  
*Start an object.*
- virtual int [end\\_object](#) ()  
*End object output (does nothing for a binary file).*
- virtual int [end\\_line](#) ()  
*End a line of output (does nothing for a binary file).*
- virtual int [init\\_file](#) ()  
*Output initialization.*
- virtual int [clean\\_up](#) ()  
*Finish the file.*

#### Protected Attributes

- bool [compressed](#)  
*True if the file is to be compressed.*
- bool [gzip](#)  
*True if the compression is to be performed by gzip.*
- std::ofstream [outs](#)  
*The output stream.*
- std::string [user\\_filename](#)  
*The filename specified by the user.*
- std::string [temp\\_filename](#)  
*The temporary filename.*

#### The output format

- int [fill](#)
- int [precision](#)

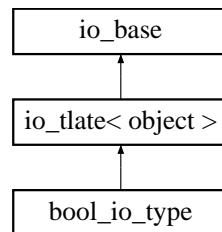
The documentation for this class was generated from the following file:

- [binary\\_file.h](#)

## 3.18 bool\_io\_type Class Reference

```
#include <collection.h>
```

Inheritance diagram for bool\_io\_type::



### 3.18.1 Detailed Description

I/O object for bool variables.

Definition at line 1642 of file collection.h.

#### Public Member Functions

- [bool\\_io\\_type](#) (const char \*t)  
*Desc.*
- [bool\\_io\\_type](#) ()
- int [addb](#) ([collection](#) &co, std::string name, bool x, bool overwrt=true)  
*Add a bool to a [collection](#).*
- bool [getb](#) ([collection](#) &co, std::string tname)  
*Get a bool from a [collection](#).*
- int [get\\_def](#) ([collection](#) &co, std::string tname, bool &op, bool def=false)  
*Get a bool from a [collection](#).*

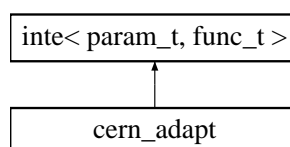
The documentation for this class was generated from the following file:

- collection.h

## 3.19 cern\_adapt Class Template Reference

```
#include <cern_adapt.h>
```

Inheritance diagram for cern\_adapt::



### 3.19.1 Detailed Description

```
template<class param_t, class func_t, size_t nsub = 100> class cern_adapt< param_t, func_t, nsub >
```

Adaptive integration (CERNLIB).

Uses [cern\\_gauss56](#) to perform adaptive integration by automatically subdividing the integration interval. At each step, the interval with the largest absolute uncertainty is divided in `o2sclf`. The routine stops if the absolute tolerance is less than `tolx`, the relative tolerance is less than `tolf`, or the number of segments exceeds the template parameter `nsub` (in which case the error handler is called, since the integration may not have been successful). The number of segments used in the last integration can be obtained from [get\\_nsegments\(\)](#).

The template parameter `nsub`, is the maximum number of subdivisions. It is automatically set to 100 in the original CERNLIB routine, and defaults to 100 here. The default base integration object is of type [cern\\_gauss56](#). This is the CERNLIB default, but can be modified by calling [set\\_inte\(\)](#).

### Todo

It looks like the first segment is of zero size, is this because there's an offset? Double check that we don't have memory issues if `nseg=nsub`.

### Idea for future

More error checking, e.g. ensure the user doesn't try to get a segment greater than the total number of segments.

### Idea for future

Allow user to set the initial segments?

Definition at line 63 of file `cern_adapt.h`.

## Public Member Functions

- [cern\\_adapt](#) ()
- `int` [set\\_inte](#) (`inte`< `param_t`, `func_t` > &`i`)  
*Set the base integration object to use.*
- `size_t` [get\\_nsegments](#) ()  
*Return the number of segments used in the last integration.*
- `int` [get\\_ith\\_segment](#) (`size_t` `i`, `double` &`xlow`, `double` &`xhigh`, `double` &`value`, `double` &`errsq`)  
*Return the `ith` segment.*
- `template`<`class` `vec_t`>  
`int` [get\\_segments](#) (`vec_t` &`xlow`, `vec_t` &`xhigh`, `vec_t` &`value`, `vec_t` &`errsq`)  
*Return all of the segments.*
- `virtual` `double` [integ](#) (`func_t` &`func`, `double` `a`, `double` `b`, `param_t` &`pa`)  
*Integrate function `func` from `a` to `b`.*
- `virtual` `int` [integ\\_err](#) (`func_t` &`func`, `double` `a`, `double` `b`, `param_t` &`pa`, `double` &`res`, `double` &`err`)  
*Integrate function `func` from `a` to `b` giving result `res` and error `err`.*

## Data Fields

- `size_t` [nseg](#)  
*Number of subdivisions.*

## Protected Attributes

- `double` [xlo](#) [`nsub`]  
*Lower end of subdivision.*
- `double` [xhi](#) [`nsub`]  
*High end of subdivision.*
- `double` [tval](#) [`nsub`]  
*Value of integral for subdivision.*
- `double` [ters](#) [`nsub`]

*Squared error for subdivision.*

- `int nter`  
*Previous number of subdivisions.*
- `cern_gauss56< param_t, func_t > cg56`  
*Default integration object.*
- `inte< param_t, func_t > * it`  
*The base integration object.*

### 3.19.2 Field Documentation

#### 3.19.2.1 size\_t nseg

Number of subdivisions.

The options are

- 0: Use previous binning and do not subdivide further
- 1: Automatic - adapt until tolerance is attained (default)
- n: (n>1) split first in n equal segments, then adapt until tolerance is obtained.

Definition at line 116 of file `cern_adapt.h`.

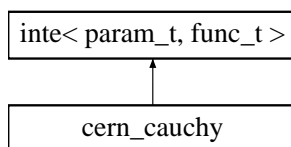
The documentation for this class was generated from the following file:

- `cern_adapt.h`

## 3.20 cern\_cauchy Class Template Reference

```
#include <cern_cauchy.h>
```

Inheritance diagram for `cern_cauchy`:



### 3.20.1 Detailed Description

```
template<class param_t, class func_t> class cern_cauchy< param_t, func_t >
```

Cauchy principal value integration (CERNLIB).

The location of the singularity must be specified before-hand in `cern_cauchy::s`, and the singularity must not be at one of the endpoints. Note that when integrating a function of the form  $\frac{f(x)}{(x-s)}$ , the denominator  $(x-s)$  must be specified in the argument `func` to `integ()`.

The method from [Longman58](#) is used for the decomposition of the integral, and the resulting integrals are computed using `cern_gauss`.

The uncertainty in the integral is not calculated, and is always given as zero. The default base integration object is of type `cern_gauss`. This is the CERNLIB default, but can be modified by calling `set_inte()`. If the singularity is outside the region of integration, then the result from the base integration object is returned without calling the error handler.

Possible errors for `integ()` and `integ_err()`:

- `gsl_inval` - Singularity is on an endpoint
- `gsl_efailed` - Couldn't reach requested accuracy

Definition at line 59 of file `cern_cauchy.h`.

### Public Member Functions

- `cern_cauchy()`
- `int set_inte(inte< param_t, func_t > &i)`  
*Set the base integration object to use.*
- `virtual int integ_err(func_t &func, double a, double b, param_t &pa, double &res, double &err)`  
*Integrate function `func` from `a` to `b` giving result `res` and error `err`.*
- `virtual double integ(func_t &func, double a, double b, param_t &pa)`  
*Integrate function `func` from `a` to `b`.*

### Data Fields

- `double s`  
*The singularity (must be set before calling `integ()` or `integ_err()`).*
- `cern_gauss< param_t, func_t > def_inte`  
*Default integration object.*

### Protected Attributes

- `inte< param_t, func_t > *it`  
*The base integration object.*

### Integration constants

- `double x[12]`
- `double w[12]`

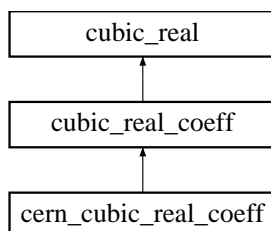
The documentation for this class was generated from the following file:

- `cern_cauchy.h`

## 3.21 cern\_cubic\_real\_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for `cern_cubic_real_coeff`:



### 3.21.1 Detailed Description

Solve a cubic with real coefficients and complex roots (CERNLIB).

Definition at line 390 of file poly.h.

#### Public Member Functions

- virtual `~cern_cubic_real_coeff()`
- virtual int `solve_rc` (const double a3, const double b3, const double c3, const double d3, double &x1, std::complex< double > &x2, std::complex< double > &x3)  
*Solves the polynomial  $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$  giving the real solution  $x = x_1$  and two complex solutions  $x = x_1, x = x_2$ , and  $x = x_3$ .*
- virtual int `rrteq3` (double r, double s, double t, double x[], double &d)  
*The original CERNLIB interface.*
- const char \* `type` ()  
*Return a string denoting the type ("cern\_cubic\_real\_coeff").*

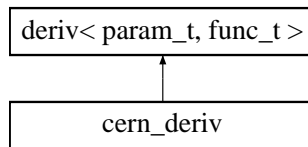
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.22 cern\_deriv Class Template Reference

```
#include <cern_deriv.h>
```

Inheritance diagram for cern\_deriv:



### 3.22.1 Detailed Description

```
template<class param_t, class func_t> class cern_deriv< param_t, func_t >
```

Numerical differentiation routine (CERNLIB).

This uses Romberg extrapolation to compute the derivative with the finite-differencing formula

$$f'(x) = [f(x+h) - f(x-h)]/(2h)$$

If `root::verbose` is greater than zero, then each iteration prints out the extrapolation [table](#), and if `root::verbose` is greater than 1, then a keypress is required at the end of each iteration.

Based on the CERNLIB routine DERIV(), which was based on [Rutishauser63](#).

#### Note:

Second and third derivatives are computed by naive nested applications of the formula for the first derivative and the uncertainty for these will likely be underestimated.

Definition at line 59 of file cern\_deriv.h.

## Public Member Functions

- `cern_deriv()`
- virtual int `calc_err` (double x, param\_t &pa, func\_t &func, double &dfdx, double &err)  
*Calculate the first derivative of func w.r.t. x and the uncertainty.*
- virtual int `calc_err_int` (double x, typename `deriv`< param\_t, func\_t >::dpars &pa, `funct`< typename `deriv`< param\_t, func\_t >::dpars > &func, double &dfdx, double &err)
- virtual const char \* `type` ()  
*Return string denoting type ("cern\_deriv").*

## Data Fields

- double `delta`  
*A scaling factor (default 1.0).*
- double `eps`  
*Extrapolation tolerance (default is  $5 \times 10^{14}$ ).*

## Protected Attributes

### Storage for the fixed coefficients

- double `dx` [10]
- double `w` [10][4]

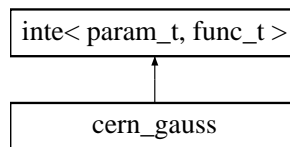
The documentation for this class was generated from the following file:

- `cern_deriv.h`

## 3.23 cern\_gauss Class Template Reference

```
#include <cern_gauss.h>
```

Inheritance diagram for `cern_gauss`:



### 3.23.1 Detailed Description

`template<class param_t, class func_t> class cern_gauss< param_t, func_t >`

Gaussian quadrature (CERNLIB).

For any interval  $(a, b)$ , we define  $g_8(a, b)$  and  $g_{16}(a, b)$  to be the 8- and 16-point Gaussian quadrature approximations to

$$I = \int_a^b f(x) dx$$

and define

$$r(a, b) = \frac{|g_{16}(a, b) - g_8(a, b)|}{1 + g_{16}(a, b)}$$



The function `integ()` returns  $G$  given by

$$G = \sum_{i=1}^k g_{16}(x_{i-1}, x_i)$$

where  $x_0 = a$  and  $x_k = b$  and the subdivision points  $x_i$  are given by

$$x_i = x_{i-1} + \lambda(B - x_{i-1})$$

where  $\lambda$  is the first number in the sequence  $1, \frac{1}{2}, \frac{1}{4}, \dots$  for which

$$r(x_{i-1}, x_i) < \text{eps}.$$

If, at any stage, the ratio

$$q = \left| \frac{x_i - x_{i-1}}{b - a} \right|$$

is so small so that  $1 + 0.005q$  is indistinguishable from unity, then the accuracy is required is not reachable and the error handler is called.

Unless there is severe cancellation, `inte::tolf` may be considered as specifying a bound on the relative error of the integral in the case that  $|I| > 1$  and an absolute error if  $|I| < 1$ . More precisely, if  $k$  is the number of subintervals from above, and if

$$I_{abs} = \int_a^b |f(x)| dx$$

then

$$\frac{|G - I|}{I_{abs} + k} < \text{tolf}$$

will nearly always be true when no error is returned. For functions with no singularities in the interval, the accuracy will usually be higher than this.

Definition at line 84 of file `cern_gauss.h`.

## Public Member Functions

- `cern_gauss()`
- virtual int `integ_err` (func\_t &func, double a, double b, param\_t &pa, double &res, double &err)  
*Integrate function func from a to b giving result res and error err.*
- virtual double `integ` (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function func from a to b.*

## Protected Attributes

### Integration constants

- double `x` [12]
- double `w` [12]

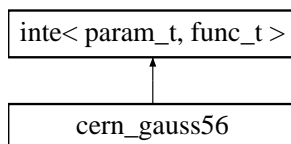
The documentation for this class was generated from the following file:

- `cern_gauss.h`

## 3.24 cern\_gauss56 Class Template Reference

```
#include <cern_gauss56.h>
```

Inheritance diagram for cern\_gauss56::



### 3.24.1 Detailed Description

```
template<class param_t, class func_t> class cern_gauss56< param_t, func_t >
```

5,6-point Gaussian quadrature (CERNLIB)

If  $I_5$  is the 5-point approximation, and  $I_6$  is the 6-point approximation to the integral, then `integ_err()` returns the result  $\frac{1}{2}(I_5 + I_6)$  with uncertainty  $|I_5 - I_6|$ .

Definition at line 41 of file cern\_gauss56.h.

#### Public Member Functions

- `cern_gauss56()`
- virtual double `integ` (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function func from a to b.*
- virtual int `integ_err` (func\_t &func, double a, double b, param\_t &pa, double &res, double &err)  
*Integrate function func from a to b giving result res and error err.*

#### Protected Attributes

##### Integration constants

- double `x5` [5]
- double `w5` [5]
- double `x6` [6]
- double `w6` [6]

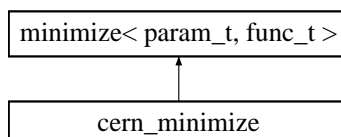
The documentation for this class was generated from the following file:

- cern\_gauss56.h

## 3.25 cern\_minimize Class Template Reference

```
#include <cern_minimize.h>
```

Inheritance diagram for cern\_minimize::



### 3.25.1 Detailed Description

**template<class param\_t, class func\_t = funct<param\_t>> class cern\_minimize< param\_t, func\_t >**

One-dimensional minimization (CERNLIB).

This is rewritten from the CERNLIB routine D503: minfc.f.

The golden section search is applied in the interval  $(a, b)$  using a fixed number  $n$  of function evaluations where

$$n = \left\lceil 2.08 \ln(|a - b|/\text{tolx}) + \frac{1}{2} \right\rceil + 1$$

The accuracy depends on the function. A choice of  $\text{tolx} > 10^{-8}$  usually results in a relative error of  $\$x\$$  which is smaller than or of the order of  $\text{tolx}$ .

This routine strictly searches the interval  $(a, b)$ . If the function is nowhere flat in this interval, then `min_bkt()` will return either  $a$  or  $b$  and `min_type` is set to 1.

#### Note:

The number of function evaluations can be quite large if `multi_min::tolx` is sufficiently small. If `multi_min::tolx` is exactly zero, then  $10^{-8}$  will be used instead.

Based on [Fletcher87](#), and [Krabs83](#).

Definition at line 60 of file `cern_minimize.h`.

#### Public Member Functions

- `cern_minimize()`
- `int nint (double x)`
- `virtual int min_bkt (double &x, double a, double b, double &y, param_t &pa, func_t &func)`  
*Calculate the minimum min of func between a and b.*
- `int set_delta (double d)`  
*Set the value of  $\delta$ .*
- `virtual const char * type ()`  
*Return string denoting type ("cern\_minimize").*

#### Data Fields

- `int min_type`  
*Type of minimum found.*

#### Protected Attributes

- `double delta`  
*The value of delta as specified by the user.*
- `bool delta_set`  
*True if the value of delta has been set.*

### 3.25.2 Member Function Documentation

**3.25.2.1 virtual int min\_bkt (double & x, double a, double b, double & y, param\_t & pa, func\_t & func)** [`inline`, `virtual`]

Calculate the minimum `min` of `func` between `a` and `b`.

The initial value of `x` is ignored.

If there is no minimum in the given interval, then on exit `x` will be equal to either `a` or `b` and `min_type` will be set to 1 instead of zero. The error handler is not called, as this need not be interpreted as an error.

Reimplemented from `minimize< param_t, func_t >`.

Definition at line 87 of file `cern_minimize.h`.

### 3.25.2.2 int set\_delta(double d) [inline]

Set the value of  $\delta$ .

If this is not called before `min_bkt()` is used, then the suggested value  $\delta = 10\text{tolx}$  is used.

Definition at line 158 of file `cern_minimize.h`.

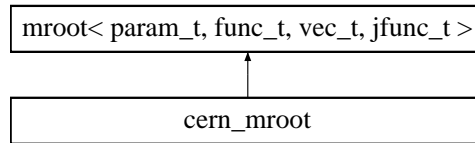
The documentation for this class was generated from the following file:

- `cern_minimize.h`

## 3.26 cern\_mroot Class Template Reference

```
#include <cern_mroot.h>
```

Inheritance diagram for `cern_mroot`:



### 3.26.1 Detailed Description

`template<class param_t, class func_t = mm_func<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class jfunc_t = jac_func<param_t,vec_t,omatrix_view>> class cern_mroot< param_t, func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t >`

Multi-dimensional mroot-finding routine (CERNLIB).

If  $x_i$  denotes the current iteration, and  $x'_i$  denotes the previous iteration, then the calculation is terminated if either of the following tests is successful

$$\begin{aligned}
 1: \quad & \max |f_i(x)| \leq \text{tolf} \\
 2: \quad & \max |x_i - x'_i| \leq \text{tolx} \times \max |x_i|
 \end{aligned}$$

This routine treats the functions specified as a `mm_func` object slightly differently than `gsl_mroot_hybrids`. First the equations should be numbered (as much as is possible) in order of increasing nonlinearity. Also, instead of calculating all of the equations on each function call, only the equation specified by the `size_t` parameter needs to be calculated. If the equations are specified as

$$\begin{aligned}
 0 &= f_0(x_0, x_1, \dots, x_{n-1}) \\
 0 &= f_1(x_0, x_1, \dots, x_{n-1}) \\
 &\dots \\
 0 &= f_{n-1}(x_0, x_1, \dots, x_{n-1})
 \end{aligned}$$

then when the `size_t` argument is given as `i`, then only the function  $f_i$  needs to be calculated.

Based on [More79](#) and [More80](#)

#### Warning:

This code has not been checked to ensure that it cannot fail to solve the equations without calling the error handler and returning a non-zero value. Until then, the solution may need to be checked explicitly by the caller.

#### Idea for future

Modify this so it handles functions which return non-zero values.

Definition at line 78 of file `cern_mroot.h`.

### Public Member Functions

- [cern\\_mroot](#) ()
- [int get\\_info](#) ()  
*Get the value of INFO from the last call to [msolve\(\)](#).*
- `virtual const char * type ()`  
*Return the type, "cern\_mroot".*
- `virtual int msolve (size_t nvar, vec_t &x, param_t &pa, func_t &func)`  
*Solve func using x as an initial guess, returning x.*

### Data Fields

- `int maxf`  
*Maximum number of function evaluations.*
- `double scale`  
*The original scale parameter from CERNLIB (default 10.0).*
- `double eps`  
*The smallest floating point number (default  $\sim 1.49012 \times 10^{-8}$ ).*

### Protected Attributes

- `alloc_t ao`  
*Memory allocator for objects of type `alloc_vec_t`.*
- `int info`  
*Internal storage for the value of `info`.*
- `int mpt [289]`  
*Store the number of function evaluations.*

## 3.26.2 Member Function Documentation

### 3.26.2.1 `int get_info ()` [inline]

Get the value of INFO from the last call to [msolve\(\)](#).

The value of `info` is assigned according to the following list. The values 1-8 are the standard behavior from CERNLIB. 0 - The function `solve()` has not been called. 1 - Test 1 was successful.

2 - Test 2 was successful.

3 - Both tests were successful.

4 - Number of iterations is greater than `cern_mroot_root::maxf`.

---

5 - Approximate (finite difference) Jacobian matrix is singular.

6 - Iterations are not making good progress.

7 - Iterations are diverging.

8 - Iterations are converging, but either `cern_mroot_root::tolx` is too small or the Jacobian is nearly singular or the variables are badly scaled.

9 - Either `root::tolf` or `root::tolx` is not greater than zero or the specified number of variables is  $\leq 0$ .

Definition at line 135 of file `cern_mroot.h`.

### 3.26.3 Field Documentation

#### 3.26.3.1 int maxf

Maximum number of function evaluations.

If `maxf`  $\leq 0$ , then  $50(nv + 3)$  (which is the CERNLIB default) is used. The default value of `maxf` is zero which then implies the default from CERNLIB.

Definition at line 144 of file `cern_mroot.h`.

#### 3.26.3.2 double eps

The smallest floating point number (default  $\sim 1.49012 \times 10^{-8}$ ).

The original prescription from CERNLIB for `eps` is given below:

```
#if !defined(CERNLIB_DOUBLE)
PARAMETER (EPS = 0.84293 69702 17878 97282 52636 392E-07)
#endif
#if defined(CERNLIB_IBM)
PARAMETER (EPS = 0.14901 16119 38476 562D-07)
#endif
#if defined(CERNLIB_VAX)
PARAMETER (EPS = 0.37252 90298 46191 40625D-08)
#endif
#if (defined(CERNLIB_UNIX)) && (defined(CERNLIB_DOUBLE))
PARAMETER (EPS = 0.14901 16119 38476 600D-07)
#endif
```

Definition at line 173 of file `cern_mroot.h`.

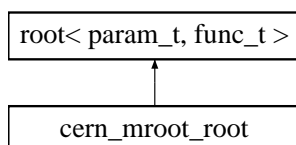
The documentation for this class was generated from the following file:

- `cern_mroot.h`

## 3.27 cern\_mroot\_root Class Template Reference

```
#include <cern_mroot_root.h>
```

Inheritance diagram for `cern_mroot_root`:



### 3.27.1 Detailed Description

**template<class param\_t, class func\_t = funct<param\_t>> class cern\_mroot\_root< param\_t, func\_t >**

One-dimensional version of [cern\\_mroot](#).

This one-dimensional root-finding routine, based on [cern\\_mroot](#), is probably slower than the more typical 1-d routines, but also tends to converge for a larger class of functions than [cern\\_root](#), [gsl\\_root\\_brent](#), or [gsl\\_root\\_stef](#). It has been modified from [cern\\_mroot](#) and slightly optimized, but has the same basic behavior.

If  $x_i$  denotes the current iteration, and  $x'_i$  denotes the previous iteration, then the calculation is terminated if either (or both) of the following tests is successful

$$\begin{aligned} 1 : \quad & \max |f_i(x)| \leq \text{tol}f \\ 2 : \quad & \max |x_i - x'_i| \leq \text{tol}x \times \max |x_i| \end{aligned}$$

#### Note:

This code has not been checked to ensure that it cannot fail to solve the equations without calling the error handler and returning a non-zero value. Until then, the solution may need to be checked explicitly by the caller.

#### Idea for future

Double-check this class to make sure it cannot fail while returning 0 for success.

Definition at line 63 of file [cern\\_mroot\\_root.h](#).

#### Public Member Functions

- [cern\\_mroot\\_root](#) ()
- virtual [~cern\\_mroot\\_root](#) ()
- int [get\\_info](#) ()  
*Get the value of INFO from the last call to [solve\(\)](#) (default 0).*
- virtual const char \* [type](#) ()  
*Return the type, "cern\_mroot\_root".*
- virtual int [solve](#) (double &ux, param\_t &pa, func\_t &func)  
*Solve func using x as an initial guess, returning x.*

#### Data Fields

- int [maxf](#)  
*Maximum number of function evaluations.*
- double [scale](#)  
*The original scale parameter from CERNLIB (default 10.0).*
- double [eps](#)  
*The smallest floating point number (default  $\sim 1.49012 \times 10^{-8}$ ).*

#### Protected Attributes

- int [info](#)  
*Internal storage for the value of info.*

### 3.27.2 Member Function Documentation

#### 3.27.2.1 int get\_info () [inline]

Get the value of INFO from the last call to [solve\(\)](#) (default 0).

The value of info is assigned according to the following list. The values 1-8 are the standard behavior from CERNLIB. 0 - The function [solve\(\)](#) has not been called. 1 - Test 1 was successful.

2 - Test 2 was successful.

3 - Both tests were successful.

4 - Number of iterations is greater than [cern\\_mroot\\_root::maxf](#).

5 - Approximate (finite difference) Jacobian matrix is singular.

6 - Iterations are not making good progress.

7 - Iterations are diverging.

8 - Iterations are converging, but either [cern\\_mroot\\_root::tolx](#) is too small or the Jacobian is nearly singular or the variables are badly scaled.

9 - Either [root::tolf](#) or [root::tolx](#) is not greater than zero.

Definition at line 96 of file [cern\\_mroot\\_root.h](#).

### 3.27.3 Field Documentation

#### 3.27.3.1 int maxf

Maximum number of function evaluations.

If  $\text{maxf} \leq 0$ , then 200 (which is the CERNLIB default) is used. The default value of `maxf` is zero which then implies the default from CERNLIB.

Definition at line 105 of file [cern\\_mroot\\_root.h](#).

#### 3.27.3.2 double eps

The smallest floating point number (default  $\sim 1.49012 \times 10^{-8}$ ).

The original prescription from CERNLIB for `eps` is given below:

```
#if !defined(CERNLIB_DOUBLE)
PARAMETER (EPS = 0.84293 69702 17878 97282 52636 392E-07)
#endif
#if defined(CERNLIB_IBM)
PARAMETER (EPS = 0.14901 16119 38476 562D-07)
#endif
#if defined(CERNLIB_VAX)
PARAMETER (EPS = 0.37252 90298 46191 40625D-08)
#endif
#if (defined(CERNLIB_UNIX)) && (defined(CERNLIB_DOUBLE))
PARAMETER (EPS = 0.14901 16119 38476 600D-07)
#endif
```

Definition at line 134 of file [cern\\_mroot\\_root.h](#).

The documentation for this class was generated from the following file:

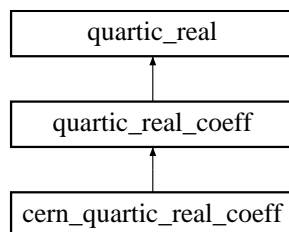
- [cern\\_mroot\\_root.h](#)



## 3.28 cern\_quartic\_real\_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for cern\_quartic\_real\_coeff:



### 3.28.1 Detailed Description

Solve a quartic with real coefficients and complex roots (CERNLIB).

Definition at line 410 of file poly.h.

#### Public Member Functions

- virtual `~cern_quartic_real_coeff()`
- virtual int `solve_rc` (const double a4, const double b4, const double c4, const double d4, const double e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)
- virtual int `rrteq4` (double a, double b, double c, double d, std::complex< double > z[], double &dc, int &mt)  
*The original CERNLIB interface.*
- const char \* `type` ()  
*Return a string denoting the type ("cern\_quartic\_real\_coeff").*

#### Protected Attributes

- `cern_cubic_real_coeff cub_obj`  
*The object to solve for the associated cubic.*

### 3.28.2 Member Function Documentation

**3.28.2.1** virtual int `solve_rc` (const double *a4*, const double *b4*, const double *c4*, const double *d4*, const double *e4*, std::complex< double > &*x1*, std::complex< double > &*x2*, std::complex< double > &*x3*, std::complex< double > &*x4*) [virtual]

Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four complex solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented from `quartic_real_coeff`.

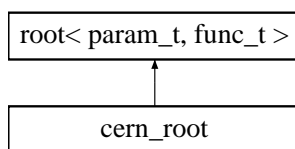
The documentation for this class was generated from the following file:

- `poly.h`

## 3.29 cern\_root Class Template Reference

```
#include <cern_root.h>
```

Inheritance diagram for cern\_root::



### 3.29.1 Detailed Description

**template<class param\_t, class func\_t = funct<param\_t>> class cern\_root< param\_t, func\_t >**

One-dimensional root-finding routine (CERNLIB).

This class attempts to find  $x_0$  and  $x_1$  in  $[a, b]$  such that  $f(x_0)f(x_1) \leq 0$ ,  $|f(x_0)| \leq |f(x_1)|$ , and  $|x_0 - x_1| \leq 2 \text{tolx} (1 + |x_0|)$ .

The variable `cern_root::tolx` defaults to  $10^{-8}$  and `cern_root::ntrial` defaults to 200.

`solve_bkt()` returns 0 for success, `gsl_einval` if the `root` is not initially bracketed, and `gsl_emaxiter` if the number of function evaluations is greater than `cern_root::ntrial`.

Based on CERNLIB routine DZEROX which is based on [Bus75](#).

Definition at line 57 of file `cern_root.h`.

#### Public Member Functions

- `cern_root ()`
- `int set_mode (int m)`  
*Set mode of solution (1 or 2).*
- `virtual const char * type ()`  
*Return the type, "cern\_root".*
- `virtual int solve_bkt (double &x1, double x2, param_t &pa, func_t &func)`  
*Solve func in region  $x_1 < x < x_2$  returning  $x_1$ .*

#### Protected Member Functions

- `double sign (double a, double b)`  
*FORTTRAN-like function for sign.*

#### Protected Attributes

- `int mode`  
*Internal storage for the mode.*

### 3.29.2 Member Function Documentation

#### 3.29.2.1 `int set_mode (int m)` [inline]

Set mode of solution (1 or 2).

- 1 should be used for simple functions where the cost is inexpensive in comparison to one iteration of `solve_bkt()`, or functions which have a pole near the `root` (this is the default).
- 2 should be used for more time-consuming functions.

If an integer other than 1 or 2 is specified, 1 is assumed.

Definition at line 107 of file cern\_root.h.

### 3.29.3 Field Documentation

#### 3.29.3.1 int mode [protected]

Internal storage for the mode.

This internal variable is actually defined to be smaller by 1 than the "mode" as it is defined in the CERNLIB documentation in order to avoid needless subtraction in [solve\\_bkt\(\)](#).

Definition at line 72 of file cern\_root.h.

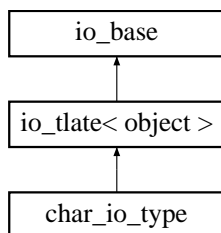
The documentation for this class was generated from the following file:

- cern\_root.h

## 3.30 char\_io\_type Class Reference

```
#include <collection.h>
```

Inheritance diagram for char\_io\_type::



### 3.30.1 Detailed Description

I/O object for char variables.

Definition at line 1674 of file collection.h.

#### Public Member Functions

- [char\\_io\\_type](#) (const char \*t)  
*Desc.*
- [char\\_io\\_type](#) ()
- int [addc](#) ([collection](#) &co, std::string name, char x, bool overwrt=true)  
*Add a char to a [collection](#).*
- char [getcc](#) ([collection](#) &co, std::string tname)  
*Get a char from a [collection](#).*
- int [get\\_def](#) ([collection](#) &co, std::string tname, char &op, char def='x')  
*Get a char from a [collection](#).*

### 3.30.2 Member Function Documentation

#### 3.30.2.1 char getcc (collection & co, std::string tname)

Get a char from a [collection](#).

Some older systems have trouble with functions named `getc`, so this is named `getc_c` instead.

The documentation for this class was generated from the following file:

- `collection.h`

## 3.31 cinput Class Reference

```
#include <collection.h>
```

### 3.31.1 Detailed Description

Class to control object input.

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

#### Public Member Functions

- `int object_in` (`std::string` type, `in_file_format` \*ins, void \*vp, `std::string` &name)  
*Input an object.*
- `int object_in` (`std::string` type, `in_file_format` \*ins, void \*vp, int sz, `std::string` &name)  
*Input an array of objects.*
- `int object_in` (`std::string` type, `in_file_format` \*ins, void \*vp, int sz, int sz2, `std::string` &name)  
*Input a 2-d array of objects.*
- `int object_in_mem` (`std::string` type, `in_file_format` \*ins, void \*&vp, `std::string` &name)  
*Input an object, allocating memory first.*
- `int object_in_mem` (`std::string` type, `in_file_format` \*ins, void \*&vp, int &sz, `std::string` &name)  
*Input an array of objects, allocating memory first.*
- `int object_in_mem` (`std::string` type, `in_file_format` \*ins, void \*&vp, int &sz, int &sz2, `std::string` &name)  
*Input a 2-d array of objects, allocating memory first.*

#### Protected Types

- `typedef std::vector< pointer_input >::iterator ipiter`  
*An iterator for the input pointers.*

#### Protected Member Functions

- `cinput` (`collection` \*co)  
*Create a new input object for a [collection](#).*
- `int assign_pointers` (`collection` \*co)  
*Assign all of the pointers read with the appropriate objects.*

#### Protected Attributes

- `std::vector< pointer_input > input_ptrs`  
*The pointers that need to be set.*
- `collection * cop`  
*The pointer to the [collection](#) stored in the constructor.*

The documentation for this class was generated from the following file:

- `collection.h`

## 3.32 cli Class Reference

```
#include <cli.h>
```

### 3.32.1 Detailed Description

Configurable command-line interface.

Default commands: help, get/set, quit, exit, '!', verbose, license, warranty, alias.

Note that if the shell command is allowed (as it is by default) there are serious security issues which arise.

Commands which begin with a '#' character are ignored.

Definition at line 205 of file cli.h.

### Public Member Functions

- int [set\\_function](#) ([comm\\_option\\_func\\_t](#) &usf)  
*Function to call when a set command is issued.*
- virtual char \* [cli\\_gets](#) (const char \*c)
- int [call\\_args](#) (std::vector< [cmd\\_line\\_arg](#) > &ca)  
*Call functions corresponding to command-line args.*
- int [process\\_args](#) (int argv, const char \*argc[ ], std::vector< [cmd\\_line\\_arg](#) > &ca, int debug=0)  
*Process command-line arguments.*
- int [process\\_args](#) (std::string s, std::vector< [cmd\\_line\\_arg](#) > &ca, int debug=0)  
*Process command-line arguments.*
- int [set\\_verbose](#) (int v)  
*Set verbosity.*
- int [run\\_interactive](#) ()  
*Run the interactive mode.*
- int [set\\_comm\\_option](#) ([comm\\_option](#) &ic)  
*Add a new command.*
- int [set\\_parameters](#) ([collection](#) &co)  
*Create a new command.*
- int [set\\_param\\_help](#) (std::string param, std::string help)  
*Set one-line help text for a parameter named param.*
- int [set\\_alias](#) (std::string alias, std::string str)  
*Set an alias alias for the string str.*

### Data Fields

- bool [gnu\\_intro](#)  
*If true, output the usual GNU intro when [run\\_interactive\(\)](#) is called.*
- bool [sync\\_verbose](#)  
*If true, then sync verbose, with a parameter of the same name.*
- bool [shell\\_cmd\\_allowed](#)  
*If true, allow the user to use ! to execute a shell command (default true).*
- std::string [prompt](#)  
*The prompt (default "> ").*
- std::string [desc](#)  
*A one- or two-line description (default is empty string).*
- std::string [cmd\\_name](#)  
*The name of the command.*
- std::string [addl\\_help\\_cmd](#)  
*Additional help text for interactive mode (default is empty string).*
- std::string [addl\\_help\\_cli](#)

*Additional help text for command-line (default is empty string).*

- char \* [line\\_read](#)

### The hard-coded command objects

- [comm\\_option c\\_help](#)
- [comm\\_option c\\_quit](#)
- [comm\\_option c\\_exit](#)
- [comm\\_option c\\_license](#)
- [comm\\_option c\\_warranty](#)
- [comm\\_option c\\_set](#)
- [comm\\_option c\\_get](#)
- [comm\\_option c\\_no\\_intro](#)
- [comm\\_option c\\_alias](#)

### Protected Member Functions

- int [apply\\_alias](#) (std::string &s, std::string sold, std::string snw)  
*Replace all occurrences of sold with snw in s.*
- int [separate](#) (std::string str, std::vector< std::string > &sv)  
*Separate a string into words.*
- bool [string\\_equal](#) (std::string s1, std::string s2)  
*Compare two strings, treating dashes as underscores.*

### The hard-coded command functions

- int [comm\\_option\\_get](#) (std::vector< std::string > &sv, bool itive\_com)
- int [comm\\_option\\_set](#) (std::vector< std::string > &sv, bool itive\_com)
- int [comm\\_option\\_help](#) (std::vector< std::string > &sv, bool itive\_com)
- int [comm\\_option\\_license](#) (std::vector< std::string > &sv, bool itive\_com)
- int [comm\\_option\\_warranty](#) (std::vector< std::string > &sv, bool itive\_com)
- int [comm\\_option\\_no\\_intro](#) (std::vector< std::string > &sv, bool itive\_com)
- int [comm\\_option\\_alias](#) (std::vector< std::string > &sv, bool itive\_com)

### Protected Attributes

- int [verbose](#)  
*Control screen output.*
- [collection](#) \* [cop](#)  
*Pointer to [collection](#) for parameters.*
- char [buf](#) [300]  
*Storage for getline.*
- [comm\\_option\\_func](#) \* [user\\_set\\_func](#)  
*Storage for the function to call after setting a parameter.*
- std::vector< [comm\\_option](#) \* > [clist](#)  
*List of commands.*

### Help for parameters

- std::vector< std::string > [ph\\_name](#)
- std::vector< std::string > [ph\\_desc](#)

### Aliases

- std::vector< std::string > [al1](#)
- std::vector< std::string > [al2](#)

### 3.32.2 Member Function Documentation

#### 3.32.2.1 int set\_verbose (int *v*)

Set verbosity.

Most errors are output to the screen even if verbose is zero.

#### 3.32.2.2 int set\_comm\_option (comm\_option & *ic*)

Add a new command.

Each command/option must have either a short form in [comm\\_option::shrt](#) or a long form in [comm\\_option::lng](#), which is unique from the other commands/options already present. You cannot add two commands/options with the same short form, even if they have different long forms, and vice versa.

#### 3.32.2.3 int set\_parameters (collection & *co*)

Create a new command.

Set the parameters with a [collection](#)

#### 3.32.2.4 int set\_alias (std::string *alias*, std::string *str*) [inline]

Set an alias *alias* for the string *str*.

Aliases can also be set using the command 'alias', but that version allows only one-word aliases.

Definition at line 381 of file cli.h.

### 3.32.3 Field Documentation

#### 3.32.3.1 bool gnu\_intro

If true, output the usual GNU intro when [run\\_interactive\(\)](#) is called.

In order to conform to GNU standards, this ought not be set to false by default.

Definition at line 270 of file cli.h.

The documentation for this class was generated from the following file:

- cli.h

## 3.33 cmd\_line\_arg Struct Reference

```
#include <cli.h>
```

### 3.33.1 Detailed Description

A command-line argument.

Definition at line 179 of file cli.h.

#### Data Fields

- std::string [arg](#)  
*The argument.*

- bool [is\\_option](#)  
*Is an option?*
- bool [is\\_valid](#)  
*Is a properly formatted option.*
- std::vector< std::string > [parms](#)  
*List of parameters (empty, unless it's an option).*
- [comm\\_option](#) \* [cop](#)  
*A pointer to the appropriate (0, unless it's an option).*

The documentation for this struct was generated from the following file:

- cli.h

## 3.34 col Struct Reference

```
#include <table.h>
```

### 3.34.1 Detailed Description

Column structure.

Definition at line 743 of file table.h.

#### Data Fields

- [ovector\\_view](#) \* [dat](#)  
*Pointer to column.*
- bool [owner](#)  
*Owner of column.*
- int [index](#)  
*Column index.*

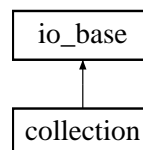
The documentation for this struct was generated from the following file:

- table.h

## 3.35 collection Class Reference

```
#include <collection.h>
```

Inheritance diagram for collection::



### 3.35.1 Detailed Description

Collection of objects.



By default, the `fout()` functions alphabetize the objects by name, but this is not a requirement for files read using `fin()`.

Important issues: 1. Pointers are not set until after an entire file is read so that objects that are pointed to may occur anywhere in a file. This means that the information that is pointed to cannot be used in the `io_tlate_d::input()` function.

### Todo

- If `pointer_in` gets a null pointer it does nothing. Should we replace this behaviour by two `pointer_in()` functions. One which does nothing if it gets a null pointer, and one which will go ahead and set the pointer to null. This is useful for output object which have default values to be used if they are given a null pointer.
- More testing on `rewrite()` function.
- Think more about adding arrays of pointers? pointers to arrays?
- Modify static data output so that if no objects of a type are included, then no static data is output for that type? (No, it's too hard to go through all objects looking for an object of a particular type).

### Bug

- Ensure that the user cannot add a object with a name of `ptrXXX`.
- `Test_type` does not test handle static data or pointers.
- Check strings and words for characters that we can't handle
- The present version of a text-file requires strings to contain at least one printable character.
- Ensure that all matching is done by both type and name if possible.

Structure: `collection::fout()` does the following:

- create an object of type `coutput`
- Add all objects in the list to the pointer map `ptr_map` (with `output=true`) so that they can be referred to by pointers later
- Output all static data using `io_type_info::static_fout()`
- Output all of the items in the list `plist` (see below). Any pointers which are not already in `ptr_map` are added at this point (with `output=false`)
- Call `coutput::pointer_map_fout()` to output all objects that were referred to but not in the list

To output individual items, `collection::fout()` does the following:

- Call either `io_base::out_wrapper()` or `io_base::out_hc_wrapper()`
- In turn, these functions call `io_base::output()`, which the user has overloaded
- If the function `io_base::output()` calls `io_tlate::object_out()` then the `io_base::output()` function appropriate for that object is called. No type or name information is included, but size integers are included if the object is a 1- or 2-d array.
- If the function `io_base::output()` calls `io_base::pointer_out()`, then the object is searched for in the `ptr_map`. If it is not there, then the object is added and assigned a name. The type and name are then output. If the pointer is `NULL`, then both the type and the name are set to `null`.

Definition at line 457 of file `collection.h`.

## Public Member Functions

- `collection ()`
- `int get_type (text_out_file &tof, std::string stype, std::string name)`  
*Output object of type stype and name name to output tof.*
- `int get (text_out_file &tof, std::string &stype, std::string name)`  
*Output object with name name to output tof.*
- `int set (std::string name, text_in_file &tif)`  
*Set object named name with input from tif.*
- `int set (std::string name, std::string val)`  
*Set object named name with input from val.*

## Output to file methods

- `int fout (out_file_format *outs)`  
*Output entire list to outs.*
- `int fout (std::string filename)`  
*Output entire list to a textfile named filename.*

## Input from file methods

If `overwrt` is true, then any objects which already exist with the same name are overwritten with the objects in the file. The `collection` owns all the objects read. (Since it created them, the `collection` assumes it ought to be responsible to destroy them.)

- `int fin (std::string file_name, bool overwrt=false, int verbose=0)`  
*Read a collection from text file named file\_name.*
- `int fin (in_file_format *ins, bool overwrt=false, int verbose=0)`  
*Read a collection from ins.*

## Miscellaneous methods

- `int test_type (o2scl::test_mgr &t, std::string stype, void *obj, void *&newobj, bool scrout=false)`  
*Test the output for type stype.*
- `int rewrite (std::string in_name, std::string out_name)`  
*Update a file containing a collection.*
- `int disown (std::string name)`  
*Force the collection to assume that the ownership of name is external.*
- `int summary (std::ostream *out, bool show_addresses=false)`  
*Summarize contents of collection.*
- `int remove (std::string name)`  
*Remove an object for the collection.*
- `void clear ()`  
*Remove all objects from the list.*
- `int npa ()`  
*Count number of objects.*

## Generic add methods

If `overwrt` is true, then any objects which already exist with the same name as `name` are overwritten. If `owner=true`, then the `collection` will own the memory allocated for the object and will free that memory with `delete` when the object is removed or the `collection` is deleted.

- `int add (std::string name, io_base *tio, void *vec, int sz=0, int sz2=0, bool overwrt=true, bool owner=false)`
- `int add (std::string name, std::string stype, void *vec, int sz=0, int sz2=0, bool overwrt=true, bool owner=false)`

## Generic get methods

- `int get (std::string tname, void *&vec)`  
*Get an object.*
- `int get (std::string tname, void *&vec, int &sz)`  
*Get an array of objects.*

- int `get` (std::string tname, void \*&vec, int &sz, int &sz2)  
*Get a 2-d array of objects.*
- int `get` (std::string tname, std::string &stype, void \*&vec)  
*Get an object and its type.*
- int `get` (std::string tname, std::string &stype, void \*&vec, int &sz)  
*Get an array of objects and their type.*
- int `get` (std::string tname, std::string &stype, void \*&vec, int &sz, int &sz2)  
*Get a 2-d array of objects and their type.*
- void \* `get` (std::string name)  
*Get an object (alternative form).*

### Input and output of individual objects

- int `out_one` (`out_file_format` \*outs, std::string stype, std::string name, void \*vp, int sz=0, int sz2=0)  
*Output one object to a file.*
- int `out_one` (std::string fname, std::string stype, std::string name, void \*vp, int sz=0, int sz2=0)  
*Output one object to a file.*
- int `in_one_name` (`in_file_format` \*ins, std::string stype, std::string name, void \*&vp, int &sz, int &sz2)  
*Input one object from a file with name name.*
- int `in_one` (`in_file_format` \*ins, std::string stype, std::string &name, void \*&vp, int &sz, int &sz2)  
*Input one object from a file.*
- int `in_one` (std::string fname, std::string stype, std::string &name, void \*&vp, int &sz, int &sz2)  
*Input one object from a file.*

### Iterator functions

- `iterator begin` ()  
*Return an [iterator](#) to the start of the [collection](#).*
- `iterator end` ()  
*Return an [iterator](#) to the end of the [collection](#).*
- `type_iterator begin` (std::string utype)  
*Return an [iterator](#) to the first element of type utype in the [collection](#).*
- `type_iterator end` (std::string utype)  
*Return an [iterator](#) to the end of the [collection](#).*

### Protected Types

- typedef std::map< std::string, `collection_entry`, `string_comp` >::iterator `piter`  
*A convenient [iterator](#) definition for the [collection](#).*

### Protected Attributes

- std::map< std::string, `collection_entry`, `string_comp` > `plist`  
*The actual [collection](#).*

### Data Structures

- class `iterator`  
*An [iterator](#) for stepping through a [collection](#).*
- class `type_iterator`  
*An [iterator](#) for stepping through the entries in a [collection](#) of a particular type.*

## 3.35.2 Member Function Documentation

### 3.35.2.1 int `rewrite` (std::string *in\_name*, std::string *out\_name*)

Update a file containing a [collection](#).

This method loads the file from "fin" and produces a file at "fout" containing all of the objects from "fin", updated by their new values in the present list if possible. Then, it adds to the end of "fout" any objects in the present list that were not originally contained in "fin".

**3.35.2.2 int disown (std::string name)**

Force the [collection](#) to assume that the ownership of `name` is external.

This allows the user to take over ownership of the object named `name`. This is particularly useful if the object is read from a file (since then object is owned initially by the [collection](#)), and you want to delete the [collection](#), but retain the object.

**3.35.2.3 int remove (std::string name)**

Remove an object for the [collection](#).

Free the memory `name` if it is owned by the [collection](#) and then remove it from the [collection](#).

**3.35.2.4 int out\_one (out\_file\_format \* outs, std::string stype, std::string name, void \* vp, int sz = 0, int sz2 = 0)**

Output one object to a file.

This does not disturb any objects in the [collection](#). The pointer specified does not need to be in the [collection](#) and is not added to the [collection](#).

**3.35.2.5 int out\_one (std::string fname, std::string stype, std::string name, void \* vp, int sz = 0, int sz2 = 0)**

Output one object to a file.

This does not disturb any objects in the [collection](#). The pointer specified does not need to be in the [collection](#) and is not added to the [collection](#).

**3.35.2.6 int in\_one\_name (in\_file\_format \* ins, std::string stype, std::string name, void \*& vp, int & sz, int & sz2)**

Input one object from a file with name `name`.

This does not disturb any objects in the [collection](#). The pointer specified does not need to be in the [collection](#) and is not added to the [collection](#).

**3.35.2.7 int in\_one (in\_file\_format \* ins, std::string stype, std::string & name, void \*& vp, int & sz, int & sz2)**

Input one object from a file.

This does not disturb any objects in the [collection](#). The pointer specified does not need to be in the [collection](#) and is not added to the [collection](#).

**3.35.2.8 int in\_one (std::string fname, std::string stype, std::string & name, void \*& vp, int & sz, int & sz2)**

Input one object from a file.

This does not disturb any objects in the [collection](#). The pointer specified does not need to be in the [collection](#) and is not added to the [collection](#).

The documentation for this class was generated from the following file:

- `collection.h`

**3.36 collection::iterator Class Reference**

```
#include <collection.h>
```

### 3.36.1 Detailed Description

An [iterator](#) for stepping through a [collection](#).

Definition at line 684 of file collection.h.

#### Public Member Functions

- [iterator operator++](#) ()  
*Prefix increment.*
- [iterator operator++](#) (int unused)  
*Postfix increment.*
- [iterator operator--](#) ()  
*Prefix decrement.*
- [collection\\_entry \\* operator →](#) () const  
*Dereference.*
- std::string [name](#) ()  
*Return the name of the [collection](#) entry.*

#### Protected Member Functions

- [iterator](#) (piter p)  
*Create an [iterator](#) from the STL [iterator](#).*

#### Protected Attributes

- [piter pit](#)  
*Local storage for the STL [iterator](#).*

#### Friends

- int [operator==](#) (const [iterator](#) &i1, const [iterator](#) &i2)  
*Equality comparison for two iterators.*
- int [operator!=](#) (const [iterator](#) &i1, const [iterator](#) &i2)  
*Inequality comparison for two iterators.*

The documentation for this class was generated from the following file:

- collection.h

## 3.37 collection::type\_iterator Class Reference

```
#include <collection.h>
```

### 3.37.1 Detailed Description

An [iterator](#) for stepping through the entries in a [collection](#) of a particular type.

Definition at line 740 of file collection.h.

---

## Public Member Functions

- `type_iterator operator++ ()`  
*Prefix increment.*
- `type_iterator operator++ (int unused)`  
*Postfix increment.*
- `collection_entry * operator → () const`  
*Dereference.*
- `std::string name ()`  
*Return the name of the `collection` entry.*

## Protected Member Functions

- `type_iterator (piter p, std::string type, collection *cop)`  
*Constructor.*

## Protected Attributes

- `std::string ltype`  
*Local storage for the type.*
- `collection * lcop`  
*Store a pointer to the `collection`.*
- `piter pit`  
*The STL `iterator`.*

## Friends

- `int operator== (const type_iterator &i1, const type_iterator &i2)`  
*Equality comparison for two iterators.*
- `int operator!= (const type_iterator &i1, const type_iterator &i2)`  
*Inequality comparison for two iterators.*

The documentation for this class was generated from the following file:

- `collection.h`

## 3.38 collection\_entry Struct Reference

```
#include <collection.h>
```

### 3.38.1 Detailed Description

An entry in a `collection`.

Definition at line 52 of file `collection.h`.

## Data Fields

- `void * data`  
*The pointer to the object.*
  - `int size`  
*The first size parameter.*
  - `int size2`
-

*The second size parameter.*

- bool `owner`  
*True if the `collection` owns this object.*
- class `io_base * iop`  
*A pointer to the corresponding `io_base` object.*

The documentation for this struct was generated from the following file:

- `collection.h`

## 3.39 columnify Class Reference

```
#include <columnify.h>
```

### 3.39.1 Detailed Description

Create nicely formatted columns from a `table` of strings.

This is a brute-force approach of order  $\text{ncols} \times \text{nrows}$ .

#### Todo

Move the `screenify()` functionality from `misc.h` into this class?

Definition at line 48 of file `columnify.h`.

### Public Member Functions

- `columnify ()`
- `template<class mat_string_t, class vec_string_t, class vec_int_t>`  
`int align (const mat_string_t &table, size_t ncols, size_t nrows, vec_string_t &ctable, vec_int_t &align_spec)`  
*Take `table` and create a new object `ctable` with appropriately formatted columns.*

### Static Public Attributes

- static const int `align_left` = 1  
*Align the left-hand sides.*
- static const int `align_right` = 2  
*Align the right-hand sides.*
- static const int `align_lmid` = 3  
*Center, slightly to the left if spacing is uneven.*
- static const int `align_rmid` = 4  
*Center, slightly to the right if spacing is uneven.*
- static const int `align_dp` = 5  
*Align with decimal points.*
- static const int `align_lnum` = 6  
*Align negative numbers to the left and use a space for positive numbers.*

### 3.39.2 Member Function Documentation

**3.39.2.1** `int align (const mat_string_t & table, size_t ncols, size_t nrows, vec_string_t & ctable, vec_int_t & align_spec)`  
[inline]

Take `table` and create a new object `ctable` with appropriately formatted columns.

---

The `table` of strings should be stored in `table` in "column-major" order, so that `table` has the interpretation of a set of columns to be aligned. Before calling `align()`, `ctable` should be allocated so that at least the first `nrows` entries can be assigned, and `align_spec` should contain `ncols` entries specifying the style of alignment for each column.

Definition at line 82 of file `columnify.h`.

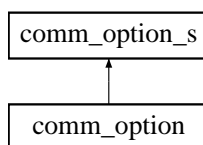
The documentation for this class was generated from the following file:

- `columnify.h`

## 3.40 comm\_option Class Reference

```
#include <cli.h>
```

Inheritance diagram for `comm_option`::



### 3.40.1 Detailed Description

Command for interactive mode in `cli`.

Definition at line 140 of file `cli.h`.

#### Public Member Functions

- `comm_option` ()
- `comm_option` (`comm_option_s` c)

#### Static Public Attributes

##### Possible values of `::type`

- static const int `command` = 0
- static const int `cl_param` = 1
- static const int `both` = 2

The documentation for this class was generated from the following file:

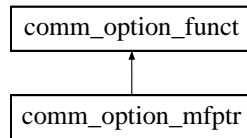
- `cli.h`

## 3.41 comm\_option\_func Class Reference

```
#include <cli.h>
```

Inheritance diagram for `comm_option_func`::





### 3.41.1 Detailed Description

Base for [cli](#) command function.

Definition at line 43 of file cli.h.

#### Public Member Functions

- [comm\\_option\\_func](#) ()
- virtual [~comm\\_option\\_func](#) ()
- virtual int [operator\(\)](#) (std::vector< std::string > &cstr, bool itive\_com)  
*The basic function called by [cli](#).*

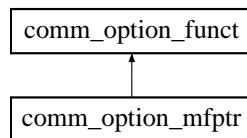
The documentation for this class was generated from the following file:

- cli.h

## 3.42 comm\_option\_mfptra Class Template Reference

```
#include <cli.h>
```

Inheritance diagram for comm\_option\_mfptra::



### 3.42.1 Detailed Description

```
template<class tclass> class comm_option_mfptra< tclass >
```

Member function pointer for [cli](#) command function.

Definition at line 67 of file cli.h.

#### Public Member Functions

- [comm\\_option\\_mfptra](#) (tclass \*tp, int(tclass::\*fp)(std::vector< std::string > &, bool))  
*Create from a member function pointer from the specified class.*
- virtual [~comm\\_option\\_mfptra](#) ()
- virtual int [operator\(\)](#) (std::vector< std::string > &cstr, bool itive\_com)  
*The basic function called by [cli](#).*

## Protected Attributes

- `int(tclass::* fptr)(std::vector< std::string > &cstr, bool itive_com)`  
The pointer to the member function.
- `tclass * tptr`  
The pointer to the class.

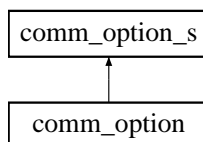
The documentation for this class was generated from the following file:

- `cli.h`

## 3.43 comm\_option\_s Struct Reference

```
#include <cli.h>
```

Inheritance diagram for `comm_option_s`:



### 3.43.1 Detailed Description

Command for interactive mode in [cli](#).

Definition at line 112 of file `cli.h`.

## Data Fields

- `char shrt`  
Short option (' \0' for none).
- `std::string lng`  
Long option (must be specified).
- `std::string desc`  
Description for help (default is empty string).
- `int min\_parms`  
Minimum number of parameters (0 for none, -1 for variable).
- `int max\_parms`  
Maximum number of parameters (0 for none, -1 for variable).
- `std::string parm\_desc`  
Description of parameters (default is empty string).
- `std::string help`  
The help description (default is empty string).
- `comm\_option\_funct * func`  
The pointer to the function to be called (or 0 for no function).
- `int type`  
Type: command-line parameter, command, or both (default command).

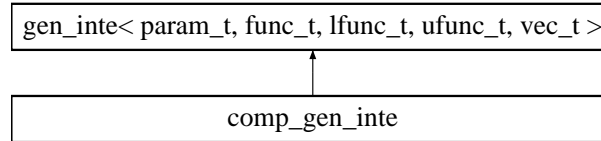
The documentation for this struct was generated from the following file:

- `cli.h`

## 3.44 comp\_gen\_inte Class Template Reference

```
#include <comp_gen_inte.h>
```

Inheritance diagram for comp\_gen\_inte::



### 3.44.1 Detailed Description

**template<class param\_t, class func\_t, class lfunc\_t = func\_t, class ufunc\_t = func\_t, class vec\_t = ovector\_view, class alloc\_vec\_t = ovector, class alloc\_t = ovector\_alloc> class comp\_gen\_inte< param\_t, func\_t, lfunc\_t, ufunc\_t, vec\_t, alloc\_vec\_t, alloc\_t >**

Naive generalized multi-dimensional integration.

Naively combine several one-dimensional integration objects from class [inte](#) in order to perform a multi-dimensional integration. The integration routines are specified in the function [set\\_ptrs\(\)](#).

The integration routines are called in order of their specification in the list [inte \\*\\*ip](#). For the example of a two-dimensional integration [ip\[0\]](#) is called first with limits [a\[0\]](#) and [b\[0\]](#) and performs the integral of the integral given by [ip\[1\]](#) of the function from [a\[1\]](#) to [b\[1\]](#), both of which may depend explicitly on [x\[0\]](#). The integral performed is:

$$\int_{x_0=a_0}^{x_0=b_0} f(x_0) \int_{x_1=a_1(x_0)}^{x_1=b_1(x_0)} f(x_0, x_1) \dots \int_{x_{n-1}=a_{n-1}(x_0, x_1, \dots, x_{n-2})}^{x_{n-1}=b_{n-1}(x_0, x_1, \dots, x_{n-2})} f(x_0, x_1, \dots, x_{n-1}) dx_{n-1} \dots dx_1 dx_0$$

See the discussion about the functions [func](#), [lower](#) and [upper](#) in the documentation for the class [gen\\_inte](#).

Definition at line 66 of file [comp\\_gen\\_inte.h](#).

### Public Member Functions

- [comp\\_gen\\_inte](#) ()
- virtual [~comp\\_gen\\_inte](#) ()
- int [set\\_ptrs](#) ([inte](#)< [od\\_parms](#), [func](#)< [od\\_parms](#) > > \*\*[ip](#), int [n](#))  
*Designate the pointers to the integration routines.*
- virtual double [ginteg](#) ([func\\_t](#) &[func](#), [size\\_t](#) [n](#), [func\\_t](#) &[lower](#), [func\\_t](#) &[upper](#), [param\\_t](#) &[pa](#))  
*Integrate function [func](#) from  $\ell_i = f_i(x_i)$  to  $u_i = g_i(x_i)$  for  $0 < i < n - 1$ .*
- virtual const char \* [type](#) ()  
*Return string denoting type ("[comp\\_gen\\_inte](#)").*

### Protected Member Functions

- double [odfunc](#) (double [x](#), [od\\_parms](#) &[od](#))  
*The one-dimensional integration function.*

### Protected Attributes

- [alloc\\_t](#) [ao](#)  
*Memory allocator for objects of type [alloc\\_vec\\_t](#).*

- [funct\\_mfptr\\_noerr](#)< [comp\\_gen\\_inte](#)< param\_t, func\_t, lfunc\_t, ufunc\_t, vec\_t, alloc\_vec\_t, alloc\_t >, [od\\_parms](#) > \* [fmn](#)  
*The function to send to the integrators.*
- [size\\_t](#) [ndim](#)  
*The number of dimensions.*
- [inte](#)< [od\\_parms](#), [funct](#)< [od\\_parms](#) > > \*\* [ptrs](#)  
*Pointers to the integration objects.*
- [bool](#) [ptrs\\_set](#)  
*True if the integration objects have been specified.*

## Data Structures

- [struct od\\_parms](#)  
*Parameters to send to the 1-d integration functions.*

### 3.44.2 Member Function Documentation

#### 3.44.2.1 int set\_ptrs (inte< od\_parms, funct< od\_parms > > \*\* ip, int n) [inline]

Designate the pointers to the integration routines.

The user can, in principle, specify the one instance of an [inte](#) object for several of the pointers, but this is discouraged as most [inte](#) objects cannot be used this way. This function will not warn you if some of the pointers specified in `ip` refer to the same object.

If more 1-d integration routines than necessary are given, the extras will be unused.

Definition at line 123 of file `comp_gen_inte.h`.

The documentation for this class was generated from the following file:

- `comp_gen_inte.h`

## 3.45 comp\_gen\_inte::od\_parms Struct Reference

```
#include <comp_gen_inte.h>
```

### 3.45.1 Detailed Description

```
template<class param_t, class func_t, class lfunc_t = func_t, class ufunc_t = func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> struct comp_gen_inte< param_t, func_t, lfunc_t, ufunc_t, vec_t, alloc_vec_t, alloc_t >::od_parms
```

Parameters to send to the 1-d integration functions.

For basic usage, the class-user needs this type to specify the parameter type for 1-d integration objects.

Definition at line 91 of file `comp_gen_inte.h`.

## Data Fields

- [alloc\\_vec\\_t](#) \* [cx](#)  
*The independent variable vector.*
- [lfunc\\_t](#) \* [lower](#)  
*The function specifying the lower limits.*
- [ufunc\\_t](#) \* [upper](#)  
*The function specifying the upper limits.*
- [func\\_t](#) \* [func](#)

*The function to be integrated.*

- int [ndim](#)  
*The number of dimensions.*
- int [idim](#)  
*The present recursion level.*
- param\_t \* [vp](#)  
*The user-specified parameter.*

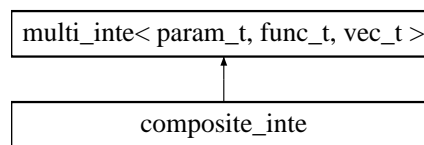
The documentation for this struct was generated from the following file:

- [comp\\_gen\\_inte.h](#)

## 3.46 composite\_inte Class Template Reference

```
#include <composite_inte.h>
```

Inheritance diagram for composite\_inte::



### 3.46.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t, class alloc\_vec\_t, class alloc\_t> class composite\_inte< param\_t, func\_t, vec\_t, alloc\_vec\_t, alloc\_t >**

Naive multi-dimensional integration over a hypercube.

Naively combine several one-dimensional integration objects from class [inte](#) in order to perform a multi-dimensional integration over a region defined by constant limits. For more general regions of integration, use children of the class [gen\\_inte](#).

The integration routines are specified in the function [set\\_ptrs\(\)](#).

The integration routines are called in order of their specification in the list [inte](#) **\*\*ip**. For the example of a two-dimensional integration **ip[0]** is called first with limits **a[0]** and **b[0]** and performs the integral of the integral given by **ip[1]** of the function from **a[1]** to **b[1]**. In other words, the integral performed is:

$$\int_{x_0=a_0}^{x_0=b_0} \int_{x_1=a_1}^{x_1=b_1} \dots \int_{x_{n-1}=a_{n-1}}^{x_{n-1}=b_{n-1}} f(x_0, x_1, \dots, x_n)$$

No error estimate is performed.

Definition at line 63 of file [composite\\_inte.h](#).

### Public Member Functions

- [composite\\_inte](#) ()
- virtual [~composite\\_inte](#) ()
- int [set\\_ptrs](#) ([inte](#)< [od\\_parms](#), [func\\_t](#)< [od\\_parms](#) > > **\*\*ip**, int n)  
*Designate the pointers to the integration routines.*
- virtual double [minteg](#) ([func\\_t](#) &**func**, size\_t n, const [vec\\_t](#) &**a**, const [vec\\_t](#) &**b**, [param\\_t](#) &**pa**)
- virtual const char \* [type](#) ()  
*Return string denoting type ("composite\_inte").*

## Protected Member Functions

- double `odfunc` (double x, `od_parms` &od)  
*The one-dimensional integration function.*

## Protected Attributes

- `alloc_t ao`  
*Memory allocator for objects of type `alloc_vec_t`.*
- `funct_mfptr_noerr< composite_inte< param_t, func_t, vec_t, alloc_vec_t, alloc_t >, od_parms > * fmn`  
*This function to send to the integrators.*
- `size_t ndim`  
*The number of dimensions.*
- `inte< od_parms, funct< od_parms > > ** iptrs`  
*Pointers to the integration objects.*
- `bool ptrs_set`  
*True if the integration objects have been specified.*

## Data Structures

- `struct od_parms`  
*Parameters to send to the 1-d integration functions.*

### 3.46.2 Member Function Documentation

#### 3.46.2.1 `int set_ptrs (inte< od_parms, funct< od_parms > > ** ip, int n) [inline]`

Designate the pointers to the integration routines.

This function allows duplicate objects in this list in order to allow the user to use only one object for more than one of the integrations.

If more 1-d integration routines than necessary are given, the extras will be unused.

Definition at line 115 of file `composite_inte.h`.

The documentation for this class was generated from the following file:

- `composite_inte.h`

## 3.47 composite\_inte::od\_parms Struct Reference

```
#include <composite_inte.h>
```

### 3.47.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t, class alloc\_vec\_t, class alloc\_t> struct composite\_inte< param\_t, func\_t, vec\_t, alloc\_vec\_t, alloc\_t >::od\_parms**

Parameters to send to the 1-d integration functions.

This structure is not intended to be frequently used directly by the class-user, but must be public so that the 1-d integration objects can be specified.

Definition at line 75 of file `composite_inte.h`.

## Data Fields

- `const vec_t * ax`  
*The user-specified upper limits.*
- `const vec_t * bx`  
*The user-specified lower limits.*
- `alloc_vec_t * cx`  
*The independent variable vector.*
- `func_t * mf`  
*The user-specified function.*
- `int ndim`  
*The user-specified number of dimensions.*
- `int idim`  
*The present recursion level.*
- `param_t * vp`  
*The user-specified parameter.*

The documentation for this struct was generated from the following file:

- `composite_inte.h`

## 3.48 contour Class Reference

```
#include <contour.h>
```

### 3.48.1 Detailed Description

Calculate [contour](#) lines from a two-dimensional data set.

#### Basic Usage

- Specify the data as a two-dimensional square grid of "heights" with [set\\_data\(\)](#).
- Specify the [contour](#) levels with [set\\_levels\(\)](#).
- Compute the contours with [calc\\_contours\(\)](#) which returns the number of contours (which is often larger than the number of [contour](#) levels, since one level can result in multiple contours).
- Retrieve the contours individually using calls to [get\\_contour\(\)](#).

The data should be stored so that the y-index is first, i.e. `data[iy][ix]`. One can always switch `x_fun` and `y_fun` if this is not the case. The data is copied by [set\\_data\(\)](#), so changing the data will not change the contours unless [set\\_data\(\)](#) is called again. The functions [set\\_levels\(\)](#) and `calc()` can be called several times for the same data without calling [set\\_data\(\)](#) again.

Linear interpolation is used to decide whether or not a line segment and a [contour](#) cross. This choice is intentional, since (in addition to making the algorithm much simpler) it is the user (and not the class) which is likely best able to refine the data. In case a simple refinement scheme is desired, the method [regrid\\_data\(\)](#) is provided which uses cubic spline interpolation to refine the data and thus make the curves more continuous.

Since linear interpolation is used, the [contour](#) calculation implicitly assumes that there is not more than one intersection of any [contour](#) level with any line segment, so if this is the case, then either a more refined data set should be specified or [regrid\\_data\(\)](#) should be used. For contours which do not close inside the region of interest, the results will always end at either the minimum or maximum values of `x_fun` or `y_fun` (no extrapolation is ever done).

As an example, for the function

$$15e^{-(x-20)^2/400-(y-5)^2/25} + 40e^{-(x-70)^2/4900-(y-2)^2/4}$$

a 10x10 grid gives the contours:

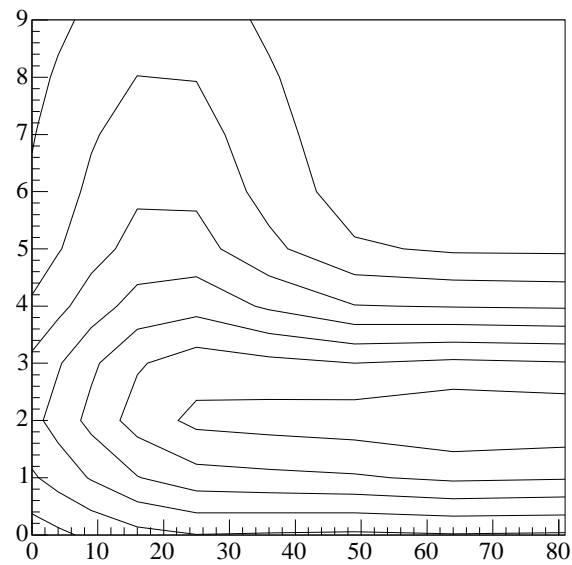


Figure 2: contourg.eps

While after a call to `regrid_data(3,3)`, the contours are a little smoother:

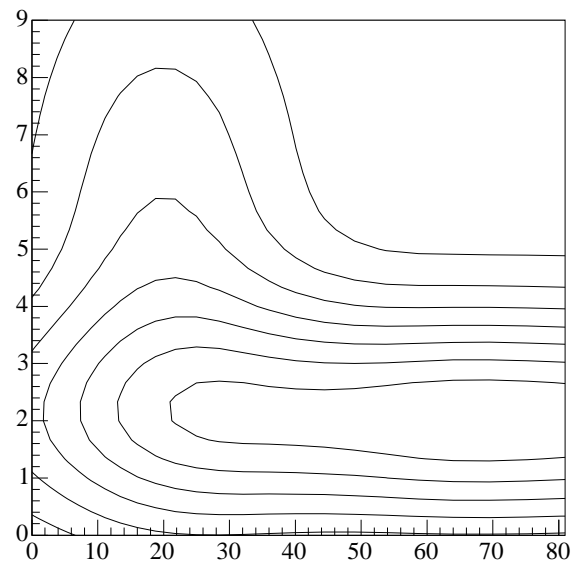


Figure 3: contourg2.eps

Mathematica gives a similar result:



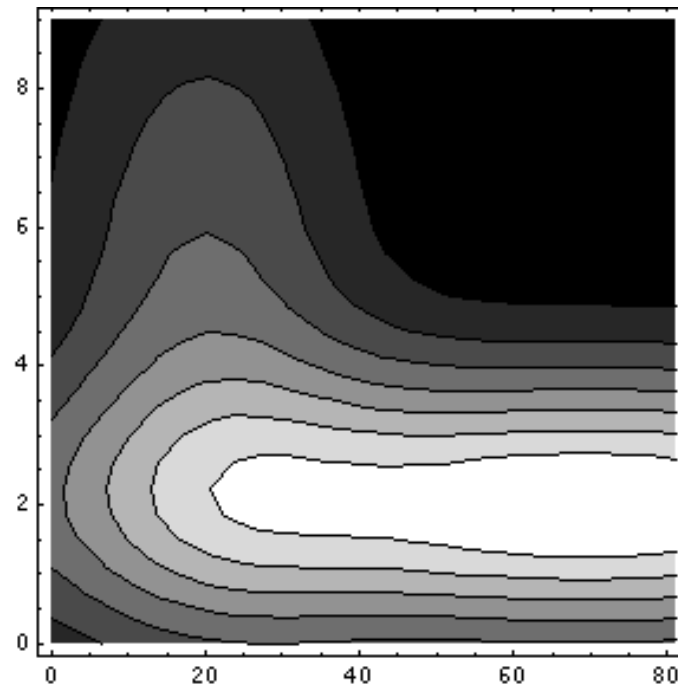


Figure 4: contour3.eps

**The Algorithm:**

This works by viewing the data as defining a square two-dimensional grid. The function `calc()` exhaustively enumerates every line segment in the grid which involves a level crossing and then organizing the points defined by the intersection of a line segment with a level curve into a full [contour](#).

**Representing Contours by Fill Regions**

If the user wants to "shade" the contours to provide a shaded or colored [contour](#) plot, then it is useful to provide a set of closed contours to be shaded instead of the (possibly) open contours returned by `calc_contours()`. After a call to `calc_contours()`, the function `regions()` can be used to organize the closed contours into regions to be shaded. Open contours are closed by adding points defining lines along the edges of the data and closed contours are inverted (if necessary) by adding an external line emanating from the closed [contour](#) and properly including the boundary edges.

**Todo**

- Some contours which should be closed are not properly closed yet. See the tests for examples which fail.
- Use `twod_intp` for `regrid_data`
- Work on how memory is allocated
- Include the functionality to provide regions to be shaded instead of just lines. This can be done by providing a method, i.e. `regions()` which converts the curves into regions.

Definition at line 126 of file `contour.h`.

**Public Member Functions****Basic usage**

- `template<class vec_t, class mat_t>`  
`int set_data (size_t sizex, size_t sizey, const vec_t &x_fun, const vec_t &y_fun, const mat_t &udata)`

*Set the data.*

- template<class vec\_t>  
int **set\_levels** (size\_t nlevels, vec\_t &ulevels)  
*Set the [contour](#) levels.*
- template<class vec\_t>  
int **calc\_contours** (vec\_t &new\_levels, bool debug=false)  
*Calculate the contours.*
- int **get\_contour\_size** (int index)  
*Return the number of points in the specified [contour](#).*
- template<class vec\_t>  
int **get\_contour** (int index, double &val, int &csize, vec\_t &x, vec\_t &y)  
*Get a [contour](#).*
- int **regions** (bool larger)  
*Compute closed [contour](#) regions (unfinished).*

### Regrid function

- int **regrid\_data** (size\_t xfact, size\_t yfact)  
*Regrid the data.*

### Obtain internal data

- int **get\_data** (size\_t &size\_x, size\_t &size\_y, const [ovector](#) \*&x\_fun, const [ovector](#) \*&y\_fun, const [ovector](#) \*\*&udata)  
*Get the data.*
- int **get\_edges** (const std::vector< [omatrix\\_int](#) \* > \*rints, const std::vector< [omatrix\\_int](#) \* > \*bints, const std::vector< [omatrix](#) \* > \*rpoints, const std::vector< [omatrix](#) \* > \*bpoints)  
*Return the edges used to compute the contours.*
- int **get\_edges\_for\_level** (size\_t nl, [omatrix\\_int](#) &rints, [omatrix\\_int](#) &bints, [omatrix](#) &rpoints, [omatrix](#) &bpoints)  
*Return the edges used to compute the contours.*

### Data Fields

- int **verbose**  
*Verbosity parameter.*
- double **lev\_adjust**  
*(default  $10^{-8}$ )*

### Protected Member Functions

- int **find\_next\_point\_right** (int j, int k, int &jnext, int &knext, int &dir\_next, int nsw=1)  
*Find next point starting from a point on a right edge.*
- int **find\_next\_point\_bottom** (int j, int k, int &jnext, int &knext, int &dir\_next, int nsw=1)  
*Find next point starting from a point on a bottom edge.*
- int **find\_intersections** (size\_t ilev, double &level)  
*Find all of the intersections of the edges with the [contour](#) level.*
- int **right\_edges** (double level, [o2scl::sm\\_interp](#) \*si)  
*Interpolate all right edge crossings.*
- int **bottom\_edges** (double level, [o2scl::sm\\_interp](#) \*si)  
*Interpolate all bottom edge crossings.*
- int **process\_line** (int j, int k, int dir, std::vector< double > &x, std::vector< double > &y, bool first=true)  
*Create a [contour](#) line from a starting edge.*
- bool **is\_point\_inside\_old** (double x1, double y1, const [ovector\\_view](#) &x, const [ovector\\_view](#) &y, double xscale=0.01, double yscale=0.01)  
*Test if a point is inside a closed [contour](#) (unfinished).*
- int **free\_memory** ()  
*Free memory.*
- bool **lines\_cross\_old** (double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)  
*Check if lines cross.*

- int `check_data` ()  
*Check to ensure the x- and y-arrays are monotonic.*
- int `smooth_contours` (size\_t nfact)  
*Smooth the contours by adding internal points using cubic interpolation (this doesn't work).*

### Protected Attributes

- int `new_debug`
- `pinside` pi  
*Object to find if a point is inside a polygon.*

### User-specified data

- int `nx`
- int `ny`
- `ovector` \* `xfun`
- `ovector` \* `yfun`
- `ovector` \*\* `data`

### User-specified contour levels

- int `nlev`
- `ovector` `levels`
- bool `levels_set`

### Generated curves

- int `ncurves`
- std::vector< int > `csizes`
- std::vector< double > `vals`
- std::vector< std::vector< double > > `xc`
- std::vector< std::vector< double > > `yc`

### Storage for edges

- std::vector< `omatrix` \* > `redges`
- std::vector< `omatrix` \* > `bedges`
- std::vector< `omatrix_int` \* > `re`
- std::vector< `omatrix_int` \* > `be`
- `omatrix_int` \* `rep`
- `omatrix_int` \* `bep`
- `omatrix` \* `redgesp`
- `omatrix` \* `bedgesp`

### Static Protected Attributes

#### Edge direction

- static const int `dright` = 0
- static const int `dbottom` = 1

#### Edge status

- static const int `empty` = 0
- static const int `edge` = 1
- static const int `contourp` = 2
- static const int `endpoint` = 3

#### Edge found or not found

- static const int `efound` = 1
- static const int `enot_found` = 0

### 3.48.2 Member Function Documentation

#### 3.48.2.1 `int set_data (size_t sizex, size_t sizey, const vec_t & x_fun, const vec_t & y_fun, const mat_t & udata)` [inline]

Set the data.

The types `vec_t` and `mat_t` can be any types which have `operator[]` and `operator[][]` for array and matrix indexing.

Note that this method copies all of the user-specified data to local storage so that changes in the data after calling this function will not be reflected in the contours that are generated.

Definition at line 147 of file `contour.h`.

#### 3.48.2.2 `int set_levels (size_t nlevels, vec_t & ulevels)` [inline]

Set the [contour](#) levels.

This is separate from the function `calc_contours()` so that the user can compute the contours for different data sets using the same levels

Definition at line 183 of file `contour.h`.

#### 3.48.2.3 `int calc_contours (vec_t & new_levels, bool debug = false)` [inline]

Calculate the contours.

The function `calc_contours()` returns the total number of contours found. Since there may be more than one disconnected contours for the same [contour](#) level, or no contours for a given level, the total number of contours may be less than or greater than the number of levels given by `set_levels()`.

If an error occurs, zero is returned.

Definition at line 205 of file `contour.h`.

#### 3.48.2.4 `int get_contour (int index, double & val, int & csize, vec_t & x, vec_t & y)` [inline]

Get a [contour](#).

Given the `index`, which is between 0 and the number of contours returned by `calc_contours()` minus 1 (inclusive), this returns the level for this [contour](#) with the x and y-values in `x` and `y` which are both of length `csize`. The vectors `x` and `y` must have been previously allocated. The user can obtain the necessary size for `x` and `y` by calling `get_contour_size()`.

Definition at line 384 of file `contour.h`.

#### 3.48.2.5 `int regrid_data (size_t xfact, size_t yfact)`

Regrid the data.

The uses cubic spline interpolation to refine the data set, ideally used before attempting to calculate the [contour](#) levels. If the original number of data points is  $(nx, ny)$ , then the new number of data points is

$$(xfact (nx - 1) + 1, yfact (ny - 1) + 1)$$

#### 3.48.2.6 `int get_data (size_t & sizex, size_t & sizey, const ovector *& x_fun, const ovector *& y_fun, const ovector **& udata)` [inline]

Get the data.

This is useful to see how the data has changed after a call to `regrid_data()`.

---

Definition at line 434 of file contour.h.

### 3.48.2.7 int get\_edges\_for\_level (size\_t nl, omatrix\_int & rints, omatrix\_int & bints, omatrix & rpoints, omatrix & bpoints)

Return the edges used to compute the contours.

This function allocates memory for `rints`, `bin``ts`, `rpoints`, and `bpoints` and fills them with a copy of the data that the class is using. As such, there is no need for them to be `const`.

### 3.48.2.8 bool is\_point\_inside\_old (double x1, double y1, const ovector\_view & x, const ovector\_view & y, double xscale = 0.01, double yscale = 0.01) [protected]

Test if a point is inside a closed [contour](#) (unfinished).

This returns true if the point (x1,y1) is "inside" the [contour](#) (i.e. a [collection](#) of line segments) given in `x` and `y`. The arrays `x` and `y` must be "ordered" so that adjacent points are placed at adjacent entries. The result is undefined if this is not the case or if the contours are not properly closed. The first and last points in `x` and `y` should be the same to indicate a closed [contour](#). The values `xscale` and `yscale` should be an approximate scale for the contours `x` and `y`.

#### Note:

This function is deprecated and has been replaced by [pinside](#)

### 3.48.2.9 bool lines\_cross\_old (double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4) [protected]

Check if lines cross.

Returns true if the line segment defined by (x1,y1) and (x2,y2) and the line segment defined by (x3,y3) and (x4,y4) have an intersection point that is between the endpoints of both segments. The function handles vertical and horizontal lines appropriately. This function will fail if `x1==y1` and `x2==y2` or if `x3==y3` and `x4==y4`, i.e. if the points do not really define a line. If the function fails, it returns false.

#### Note:

This function is deprecated and has been replaced by [pinside](#)

### 3.48.2.10 int smooth\_contours (size\_t nfact) [protected]

Smooth the contours by adding internal points using cubic interpolation (this doesn't work).

This makes the contours smoother by adding internal points between each [contour](#) line segment determined by cubic spline interpolation.

For more accurate contours, it is better to provide the original data on a finer grid, or use [regrid\\_data\(\)](#).

The documentation for this class was generated from the following file:

- contour.h

## 3.49 coutput Class Reference

```
#include <collection.h>
```

---

### 3.49.1 Detailed Description

Class to control object output.

Definition at line 915 of file collection.h.

#### Public Member Functions

- int [object\\_out](#) (std::string type, [out\\_file\\_format](#) \*outs, void \*op, int sz=0, int sz2=0, std::string name="")  
*Output an object.*

#### Protected Types

- typedef std::map< void \*, [pointer\\_output](#), [ltptr](#) >::iterator [pmiter](#)  
*A convenient iterator for the pointer list.*

#### Protected Member Functions

- [coutput](#) (class [collection](#) \*co)  
*Create a new object from a pointer to a [collection](#).*
- int [pointer\\_lookup](#) (void \*vp, std::string &name, [collection\\_entry](#) \*&ep)  
*Look for an object in the [collection](#) given a pointer.*
- int [pointer\\_map\\_fout](#) ([out\\_file\\_format](#) \*out)  
*Output all of the remaining pointers to 'out'.*

#### Protected Attributes

- std::map< void \*, [pointer\\_output](#), [ltptr](#) > [ptr\\_map](#)  
*The list pointers to object to be written to the file.*
- [collection](#) \* [cop](#)  
*The pointer to the [collection](#) stored in the constructor.*
- int [npointers](#)  
*Keep track of the number of pointers added to [ptr\\_map](#).*

#### Data Structures

- struct [ltptr](#)  
*Order the pointers by numeric value.*

### 3.49.2 Member Function Documentation

#### 3.49.2.1 int [pointer\\_lookup](#) (void \* *vp*, std::string & *name*, [collection\\_entry](#) \*& *ep*) [protected]

Look for an object in the [collection](#) given a pointer.

Lookup the pointer *vp* in the [collection](#), and return its name and [collection\\_entry](#)

### 3.49.3 Field Documentation

#### 3.49.3.1 int [npointers](#) [protected]

Keep track of the number of pointers added to [ptr\\_map](#).

These are counted for the purposes of making a unique name. This is initialized in [fout\(\)](#) and incremented in [io\\_base::pointer\\_out](#)

Definition at line 971 of file collection.h.

The documentation for this class was generated from the following file:

- collection.h

## 3.50 coutput::ltptr Struct Reference

```
#include <collection.h>
```

### 3.50.1 Detailed Description

Order the pointers by numeric value.

Definition at line 937 of file collection.h.

#### Public Member Functions

- bool [operator\(\)](#) (const void \*p1, const void \*p2) const  
*Returns  $p_1 < p_2$ .*

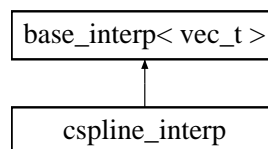
The documentation for this struct was generated from the following file:

- collection.h

## 3.51 cspline\_interp Class Template Reference

```
#include <interp.h>
```

Inheritance diagram for cspline\_interp::



### 3.51.1 Detailed Description

```
template<class vec_t> class cspline_interp< vec_t >
```

Cubic spline interpolation (GSL).

Definition at line 254 of file interp.h.

#### Public Member Functions

- [cspline\\_interp](#) (bool periodic=false)  
*Create a base interpolation object with natural or periodic boundary conditions.*
- virtual [~cspline\\_interp](#) ()
- virtual int [alloc](#) (size\_t size)  
*Allocate memory, assuming x and y have size size.*

- virtual int [init](#) (const vec\_t &xa, const vec\_t &ya, size\_t size)  
*Initialize interpolation routine.*
- virtual int [free](#) ()  
*Free allocated memory.*
- virtual int [interp](#) (const vec\_t &x\_array, const vec\_t &y\_array, size\_t size, double x, double &y)  
*Give the value of the function  $y(x = x_0)$ .*
- virtual int [deriv](#) (const vec\_t &x\_array, const vec\_t &y\_array, size\_t size, double x, double &dydx)  
*Give the value of the derivative  $y'(x = x_0)$ .*
- virtual int [deriv2](#) (const vec\_t &x\_array, const vec\_t &y\_array, size\_t size, double x, double &d2ydx2)  
*Give the value of the second derivative  $y''(x = x_0)$ .*
- virtual int [integ](#) (const vec\_t &x\_array, const vec\_t &y\_array, size\_t size, double a, double b, double &result)  
*Give the value of the integral  $\int_a^b y(x) dx$ .*

### Protected Member Functions

- void [coeff\\_calc](#) (const double c\_array[ ], double dy, double dx, size\_t index, double \*b, double \*c2, double \*d)  
*Compute coefficients for cubic spline interpolation.*
- double [integ\\_eval](#) (double ai, double bi, double ci, double di, double xi, double a, double b)  
*An internal function to assist in computing the integral.*

### Protected Attributes

- bool [peri](#)  
*True for periodic boundary conditions.*

### Storage for cubic spline interpolation

- double \* [c](#)
- double \* [g](#)
- double \* [diag](#)
- double \* [offdiag](#)

## 3.51.2 Member Function Documentation

### 3.51.2.1 virtual int init (const vec\_t & xa, const vec\_t & ya, size\_t size) [inline, virtual]

Initialize interpolation routine.

Periodic boundary conditions

Natural boundary conditions

Reimplemented from [base\\_interp](#).

Definition at line 339 of file `interp.h`.

The documentation for this class was generated from the following file:

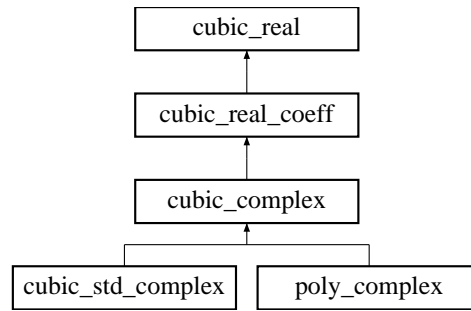
- `interp.h`

## 3.52 cubic\_complex Class Reference

```
#include <poly.h>
```

Inheritance diagram for `cubic_complex`:





### 3.52.1 Detailed Description

Solve a cubic polynomial with complex coefficients and complex roots.

Definition at line 188 of file `poly.h`.

#### Public Member Functions

- virtual `~cubic_complex()`
- virtual int `solve_r` (const double a3, const double b3, const double c3, const double d3, double &x1, double &x2, double &x3)  
*Solves the polynomial  $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$  giving the three solutions  $x = x_1$ ,  $x = x_2$ , and  $x = x_3$ .*
- virtual int `solve_rc` (const double a3, const double b3, const double c3, const double d3, double &x1, std::complex< double > &x2, std::complex< double > &x3)  
*Solves the polynomial  $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$  giving the real solution  $x = x_1$  and two complex solutions  $x = x_1$ ,  $x = x_2$ , and  $x = x_3$ .*
- virtual int `solve_c` (const std::complex< double > a3, const std::complex< double > b3, const std::complex< double > c3, const std::complex< double > d3, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3)
- const char \* `type` ()  
*Return a string denoting the type ("cubic\_complex").*

### 3.52.2 Member Function Documentation

**3.52.2.1** virtual int `solve_c` (const std::complex< double > a3, const std::complex< double > b3, const std::complex< double > c3, const std::complex< double > d3, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3) [inline, virtual]

Solves the complex polynomial  $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$  giving the three complex solutions  $x = x_1$ ,  $x = x_2$ , and  $x = x_3$ .

Reimplemented in `cubic_std_complex`.

Definition at line 215 of file `poly.h`.

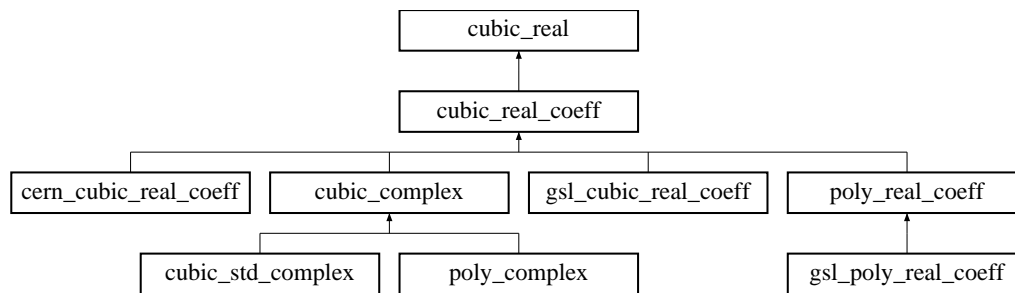
The documentation for this class was generated from the following file:

- `poly.h`

## 3.53 cubic\_real Class Reference

```
#include <poly.h>
```

Inheritance diagram for `cubic_real`:



### 3.53.1 Detailed Description

Solve a cubic polynomial with real coefficients and real roots.

Definition at line 138 of file `poly.h`.

#### Public Member Functions

- virtual `~cubic_real()`
- virtual `int solve_r(const double a3, const double b3, const double c3, const double d3, double &x1, double &x2, double &x3)`  
*Solves the polynomial  $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$  giving the three solutions  $x = x_1$ ,  $x = x_2$ , and  $x = x_3$ .*
- `const char * type()`  
*Return a string denoting the type ("cubic\_real").*

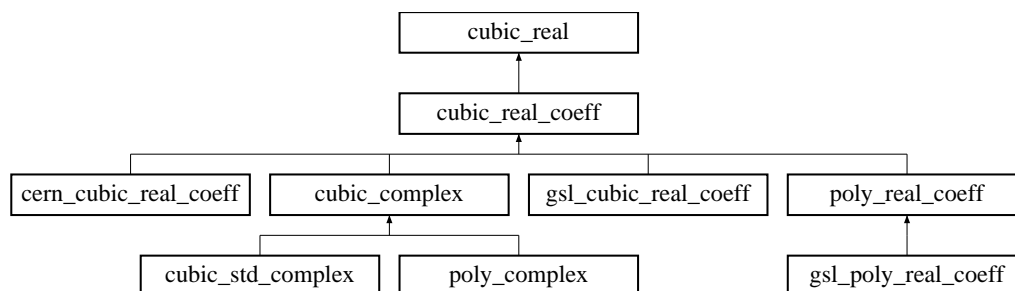
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.54 `cubic_real_coeff` Class Reference

```
#include <poly.h>
```

Inheritance diagram for `cubic_real_coeff`:



### 3.54.1 Detailed Description

Solve a cubic polynomial with real coefficients and complex roots.

Definition at line 158 of file `poly.h`.

## Public Member Functions

- virtual [~cubic\\_real\\_coeff](#) ()
- virtual int [solve\\_r](#) (const double a3, const double b3, const double c3, const double d3, double &x1, double &x2, double &x3)  
*Solves the polynomial  $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$  giving the three solutions  $x = x_1$ ,  $x = x_2$ , and  $x = x_3$ .*
- virtual int [solve\\_rc](#) (const double a3, const double b3, const double c3, const double d3, double &x1, std::complex< double > &x2, std::complex< double > &x3)  
*Solves the polynomial  $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$  giving the real solution  $x = x_1$  and two complex solutions  $x = x_1$ ,  $x = x_2$ , and  $x = x_3$ .*
- const char \* [type](#) ()  
*Return a string denoting the type ("cubic\_real\_coeff").*

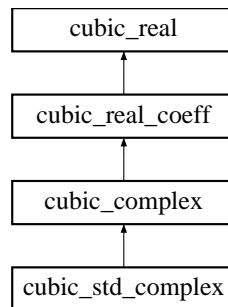
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.55 cubic\_std\_complex Class Reference

```
#include <poly.h>
```

Inheritance diagram for cubic\_std\_complex::



### 3.55.1 Detailed Description

Solve a cubic with complex coefficients and complex roots.

Definition at line 610 of file poly.h.

## Public Member Functions

- virtual [~cubic\\_std\\_complex](#) ()
- virtual int [solve\\_c](#) (const std::complex< double > a3, const std::complex< double > b3, const std::complex< double > c3, const std::complex< double > d3, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3)  
*Return a string denoting the type ("cubic\_std\_complex").*

### 3.55.2 Member Function Documentation

**3.55.2.1** virtual int [solve\\_c](#) (const std::complex< double > a3, const std::complex< double > b3, const std::complex< double > c3, const std::complex< double > d3, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3) [virtual]

Solves the complex polynomial  $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$  giving the three complex solutions  $x = x_1$ ,  $x = x_2$ , and  $x = x_3$ .

Reimplemented from [cubic\\_complex](#).

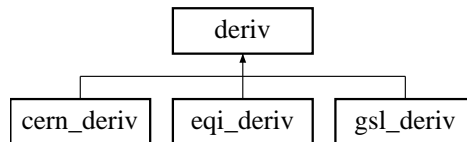
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.56 deriv Class Template Reference

```
#include <deriv.h>
```

Inheritance diagram for deriv::



### 3.56.1 Detailed Description

```
template<class param_t, class func_t> class deriv< param_t, func_t >
```

Numerical differentiation base.

This base class does not perform any actual differentiation. Use one of the children [cern\\_deriv](#), [gsl\\_deriv](#), or [eqi\\_deriv](#) instead.

This base class contains some code to automatically apply the first derivative routines to compute second or third derivatives. The error estimates for these will likely be underestimated.

#### Note:

Because this class template aims to automatically provide second and third derivatives, one must overload either both [calc\(\)](#) and [calc\\_int\(\)](#) or both [calc\\_err\(\)](#) and [calc\\_err\\_int\(\)](#).

#### Idea for future

Improve the methods for second and third derivatives

Definition at line 54 of file [deriv.h](#).

### Public Member Functions

- [deriv](#) ()
- virtual [~deriv](#) ()
- virtual double [calc](#) (double x, param\_t &pa, func\_t &func)  
*Calculate the first derivative of func w.r.t. x.*
- virtual double [calc2](#) (double x, param\_t &pa, func\_t &func)  
*Calculate the second derivative of func w.r.t. x.*
- virtual double [calc3](#) (double x, param\_t &pa, func\_t &func)  
*Calculate the third derivative of func w.r.t. x.*
- virtual double [get\\_err](#) ()  
*Get uncertainty of last calculation.*
- virtual int [calc\\_err](#) (double x, param\_t &pa, func\_t &func, double &dfdx, double &err)  
*Calculate the first derivative of func w.r.t. x and the uncertainty.*

- virtual int `calc2_err` (double x, param\_t &pa, func\_t &func, double &d2fdx2, double &err)  
*Calculate the second derivative of func w.r.t. x and the uncertainty.*
- virtual int `calc3_err` (double x, param\_t &pa, func\_t &func, double &d3fdx3, double &err)  
*Calculate the third derivative of func w.r.t. x and the uncertainty.*
- virtual const char \* `type` ()  
*Return string denoting type ("deriv").*

### Data Fields

- int `verbose`  
*Output control.*

### Protected Member Functions

- virtual double `calc_int` (double x, dpars &pa, o2scl::funct< dpars > &func)  
*Calculate the first derivative of func w.r.t. x.*
- virtual int `calc_err_int` (double x, dpars &pa, o2scl::funct< dpars > &func, double &dfdx, double &err)  
*Calculate the first derivative of func w.r.t. x and the uncertainty.*
- double `derivfun` (double x, dpars &dp)  
*The function for the second derivative.*
- double `derivfun2` (double x, dpars &dp)  
*The function for the third derivative.*

### Protected Attributes

- bool `from_calc`  
*Avoids infinite loops in case the user calls the base class version.*
- double `derr`  
*The uncertainty in the most recent derivative computation.*

### Data Structures

- struct `dpars`  
*A structure for passing the function to second and third derivatives.*

## 3.56.2 Member Function Documentation

### 3.56.2.1 virtual double calc (double x, param\_t &pa, func\_t &func) [inline, virtual]

Calculate the first derivative of func w.r.t. x.

After calling `calc()`, the error may be obtained from `get_err()`.

Definition at line 89 of file deriv.h.

### 3.56.2.2 virtual double calc\_int (double x, dpars &pa, o2scl::funct< dpars > &func) [inline, protected, virtual]

Calculate the first derivative of func w.r.t. x.

This is an internal version of `calc()` which is used in computing second and third derivatives

Definition at line 177 of file deriv.h.

**3.56.2.3 virtual int calc\_err\_int** (double *x*, dpars & *pa*, o2scl::funct< dpars > & *func*, double & *dfdx*, double & *err*)  
 [inline, protected, virtual]

Calculate the first derivative of *func* w.r.t. *x* and the uncertainty.

This is an internal version of [calc\\_err\(\)](#) which is used in computing second and third derivatives

Definition at line 191 of file `deriv.h`.

The documentation for this class was generated from the following file:

- `deriv.h`

## 3.57 deriv::dpars Struct Reference

```
#include <deriv.h>
```

### 3.57.1 Detailed Description

**template<class param\_t, class func\_t> struct deriv< param\_t, func\_t >::dpars**

A structure for passing the function to second and third derivatives.

Definition at line 61 of file `deriv.h`.

#### Data Fields

- `func_t * func`  
*The pointer to the function.*
- `param_t * up`  
*The pointer to the user-specified parameters.*

The documentation for this struct was generated from the following file:

- `deriv.h`

## 3.58 deriv\_ioc Class Reference

```
#include <deriv_ioc.h>
```

### 3.58.1 Detailed Description

Setup I/O objects for numerical differentiation classes.

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

#### Public Member Functions

- [deriv\\_ioc\(\)](#)
- [~deriv\\_ioc\(\)](#)

**Data Fields**

- deriv\_io\_type \* [deriv\\_io](#)
- eqi\_deriv\_io\_type \* [eqi\\_deriv\\_io](#)
- cern\_deriv\_io\_type \* [cern\\_deriv\\_io](#)
- gsl\_deriv\_io\_type \* [gsl\\_deriv\\_io](#)

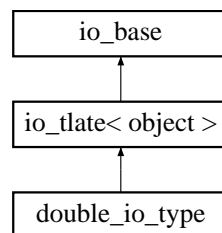
The documentation for this class was generated from the following file:

- deriv\_ioc.h

**3.59 double\_io\_type Class Reference**

```
#include <collection.h>
```

Inheritance diagram for double\_io\_type::

**3.59.1 Detailed Description**

I/O object for double variables.

Definition at line 1710 of file collection.h.

**Public Member Functions**

- [double\\_io\\_type](#) (const char \*t)  
*Desc.*
- [double\\_io\\_type](#) ()
- int [add](#) ([collection](#) &co, std::string name, double x, bool overwrt=true)  
*Add a double to a [collection](#).*
- double [getd](#) ([collection](#) &co, std::string tname)  
*Get a double from a [collection](#).*
- int [get\\_def](#) ([collection](#) &co, std::string tname, double &op, double def=0.0)  
*Get a double from a [collection](#).*

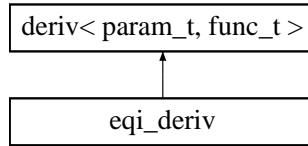
The documentation for this class was generated from the following file:

- collection.h

**3.60 eqi\_deriv Class Template Reference**

```
#include <eqi_deriv.h>
```

Inheritance diagram for eqi\_deriv::



### 3.60.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t = ovector\_view> class eqi\_deriv< param\_t, func\_t, vec\_t >**

Derivatives for equally-spaced abscissas.

This is an implementation of the formulas for equally-spaced abscissas as indicated below. The level of approximation is specified in [set\\_npoints\(\)](#). The value of  $p \times h$  can be specified in `xoфф` (default is zero).

#### Note:

The derivatives given, for example, from the five-point formula can sometimes be more accurate than computing the derivative from the interpolation class. This is especially true near the boundaries of the interpolated region.

#### Todo

The uncertainties in the derivatives are not yet computed and the second and third derivative formulas are not yet finished.

Two-point formula (note that this is independent of p).

$$f'(x_0 + ph) = \frac{1}{h} [f_1 - f_0]$$

Three-point formula from Abramowitz and Stegun

$$f'(x_0 + ph) = \frac{1}{h} \left[ \frac{2p-1}{2} f_{-1} - 2p f_0 + \frac{2p+1}{2} f_1 \right]$$

Four-point formula from Abramowitz and Stegun

$$f'(x_0 + ph) = \frac{1}{h} \left[ -\frac{3p^2 - 6p + 2}{6} f_{-1} + \frac{3p^2 - 4p - 1}{2} f_0 - \frac{3p^2 - 2p - 2}{2} f_1 + \frac{3p^2 - 1}{6} f_2 \right]$$

Five-point formula from Abramowitz and Stegun

$$\begin{aligned} f'(x_0 + ph) = & \frac{1}{h} \left[ \frac{2p^3 - 3p^2 - p + 1}{12} f_{-2} - \frac{4p^3 - 3p^2 - 8p + 4}{6} f_{-1} \right. \\ & + \frac{2p^3 - 5p}{2} f_0 - \frac{4p^3 + 3p^2 - 8p - 4}{6} f_1 \\ & \left. + \frac{2p^3 + 3p^2 - p - 1}{12} f_2 \right] \end{aligned}$$

The relations above can be confined to give formulas for second derivative formulas: Three-point formula

$$f''(x_0 + ph) = \frac{1}{h^2} [f_{-1} - 2f_0 + f_1]$$

Four-point formula:

$$f''(x_0 + ph) = \frac{1}{2h^2} [(1 - 2p) f_{-1} - (1 - 6p) f_0 - (1 + 6p) f_1 + (1 + 2p) f_2]$$

Five-point formula:

$$f''(x_0 + ph) = \frac{1}{4h^2} \left[ (1 - 2p)^2 f_{-2} + (8p - 16p^2) f_{-1} - (2 - 24p^2) f_0 - (8p + 16p^2) f_1 + (1 + 2p)^2 f_2 \right]$$



Six-point formula:

$$\begin{aligned} f'(x_0 + ph) = & \frac{1}{12h^2} [(2 - 10p + 15p^2 - 6p^3) f_{-2} + (3 + 14p - 57p^2 + 30p^3) f_{-1} \\ & + (-8 + 20p + 78p^2 - 60p^3) f_0 + (-2 - 44p - 42p^2 + 60p^3) f_1 \\ & + (6 + 22p + 3p^2 - 30p^3) f_2 + (-1 - 2p + 3p^2 + 6p^3) f_3] \end{aligned}$$

Seven-point formula:

$$\begin{aligned} f'(x_0 + ph) = & \frac{1}{36h^2} [(4 - 24p + 48p^2 - 36p^3 + 9p^4) f_{-3} + (12 + 12p - 162p^2 + 180p^3 - 54p^4) f_{-2} \\ & + (-15 + 120p + 162p^2 - 360p^3 + 135p^4) f_{-1} - 4(8 + 48p - 3p^2 - 90p^3 + 45p^4) f_0 \\ & + 3(14 + 32p - 36p^2 - 60p^3 + 45p^4) f_1 + (-12 - 12p + 54p^2 + 36p^3 - 54p^4) f_2 \\ & + (1 - 6p^2 + 9p^4) f_3] \end{aligned}$$

Definition at line 135 of file eqi\_deriv.h.

## Public Member Functions

- [eqi\\_deriv](#) ()
- [int set\\_npoints](#) (int npoints)  
*Set the number of points to use for first derivatives (default 5).*
- [int set\\_npoints2](#) (int npoints)  
*Set the number of points to use for second derivatives (default 5).*
- [virtual double calc](#) (double x, void \*pa, func\_t &func)  
*Calculate the first derivative of func w.r.t. x.*
- [virtual double calc2](#) (double x, void \*pa, func\_t &func)  
*Calculate the second derivative of func w.r.t. x.*
- [virtual double calc3](#) (double x, void \*pa, func\_t &func)  
*Calculate the third derivative of func w.r.t. x.*
- [double calc\\_array](#) (double x, double x0, double dx, size\_t nx, const vec\_t &y)  
*Calculate the derivative at x given an array.*
- [double calc2\\_array](#) (double x, double x0, double dx, size\_t nx, const vec\_t &y)  
*Calculate the second derivative at x given an array.*
- [double calc3\\_array](#) (double x, double x0, double dx, size\_t nx, const vec\_t &y)  
*Calculate the third derivative at x given an array.*
- [int deriv\\_array](#) (size\_t nv, double dx, const vec\_t &y, vec\_t &dydx)  
*Calculate the derivative of an entire array.*
- [virtual const char \\* type](#) ()  
*Return string denoting type ("eqi\_deriv").*

## Data Fields

- [double h](#)  
*Stepsize (Default  $10^{-4}$ ).*
- [double xoff](#)  
*Offset (default 0.0).*

## 3.60.2 Member Function Documentation

### 3.60.2.1 int set\_npoints (int npoints) [inline]

Set the number of points to use for first derivatives (default 5).

Acceptable values are 2-5 (see above).

Definition at line 157 of file eqi\_deriv.h.

**3.60.2.2 int set\_npoints2 (int npoints) [inline]**

Set the number of points to use for second derivatives (default 5).

Acceptable values are 3-5 (see above).

Definition at line 183 of file eqi\_deriv.h.

**3.60.2.3 double calc\_array (double x, double x0, double dx, size\_t nx, const vec\_t & y) [inline]**

Calculate the derivative at  $x$  given an array.

This calculates the derivative at  $x$  given a func\_tion specified in an array  $y$  of size  $nx$  with equally spaced abscissas. The first abscissa should be given as  $x_0$  and the distance between adjacent abscissas should be given as  $dx$ . The value  $x$  need not be one of the abscissas (i.e. it can lie in between an interval). The appropriate offset is calculated automatically.

Definition at line 234 of file eqi\_deriv.h.

**3.60.2.4 double calc2\_array (double x, double x0, double dx, size\_t nx, const vec\_t & y) [inline]**

Calculate the second derivative at  $x$  given an array.

This calculates the second derivative at  $x$  given a func\_tion specified in an array  $y$  of size  $nx$  with equally spaced abscissas. The first abscissa should be given as  $x_0$  and the distance between adjacent abscissas should be given as  $dx$ . The value  $x$  need not be one of the abscissas (i.e. it can lie in between an interval). The appropriate offset is calculated automatically.

Definition at line 251 of file eqi\_deriv.h.

**3.60.2.5 double calc3\_array (double x, double x0, double dx, size\_t nx, const vec\_t & y) [inline]**

Calculate the third derivative at  $x$  given an array.

This calculates the third derivative at  $x$  given a function specified in an array  $y$  of size  $nx$  with equally spaced abscissas. The first abscissa should be given as  $x_0$  and the distance between adjacent abscissas should be given as  $dx$ . The value  $x$  need not be one of the abscissas (i.e. it can lie in between an interval). The appropriate offset is calculated automatically.

Definition at line 269 of file eqi\_deriv.h.

**3.60.2.6 int deriv\_array (size\_t nv, double dx, const vec\_t & y, vec\_t & dydx) [inline]**

Calculate the derivative of an entire array.

Right now this uses  $np=5$ .

**Todo**

generalize to other values of npoints.

Definition at line 283 of file eqi\_deriv.h.

The documentation for this class was generated from the following file:

- eqi\_deriv.h

**3.61 err\_class Class Reference**

```
#include <err_hnd.h>
```

### 3.61.1 Detailed Description

The error handler.

An error handler for use in O2scl which replaces the GSL error handler

Note that the string arguments to [set\(\)](#) can refer to temporary storage, since they are copied when the function is called and an error is set.

Definition at line 135 of file `err_hnd.h`.

#### Public Member Functions

- void [set](#) (const char \*reason, const char \*file, int line, int lerrno)  
*Set an error.*
- void [add](#) (const char \*reason, const char \*file, int line, int lerrno)  
*Add information to previous error.*
- void [get](#) (const char \*&reason, const char \*&file, int &line, int &lerrno)  
*Get the last error.*
- int [get\\_errno](#) ()  
*Return the last error number.*
- int [get\\_line](#) ()  
*Return the line number of the last error.*
- const char \* [get\\_reason](#) ()  
*Return the reason for the last error.*
- const char \* [get\\_file](#) ()  
*Return the file name of the last error.*
- const char \* [get\\_str](#) ()  
*Return a string summarizing the last error.*
- void [reset](#) ()  
*Remove last error information.*
- void [set\\_mode](#) (int m)  
*Force a hard exit if an error occurs.*

#### Static Public Member Functions

- static void [gsl\\_hnd](#) (const char \*reason, const char \*file, int line, int lerrno)  
*Set an error.*

#### Data Fields

- bool [array\\_abort](#)  
*If true, call `exit()` when an array index error is set (default true).*

#### Protected Attributes

- int [a\\_errno](#)  
*The error number.*
- int [a\\_line](#)  
*The line number.*
- int [mode](#)  
*The mode of error handling.*
- char \* [a\\_file](#)  
*The filename.*
- char [a\\_reason](#) [[rsize](#)]  
*The error explanation.*
- char [fullstr](#) [[fsize](#)]  
*A full string with explanation and line and file info.*

## Static Protected Attributes

- static `err_class * ptr`  
*A pointer to the default error handler.*
- static const int `rsize = 300`  
*The maximum size of error explanations.*
- static const int `fsize = 400`  
*The maximum size of error explanations with the line and file info.*

### 3.61.2 Member Function Documentation

#### 3.61.2.1 static void `gsl_hnd (const char * reason, const char * file, int line, int lerrno)` [static]

Set an error.

This is separate from `set()`, since the `gsl` error handler needs to be a static function.

### 3.61.3 Field Documentation

#### 3.61.3.1 bool `array_abort`

If true, call `exit()` when an array index error is set (default true).

This is ignored if `O2SCL_ARRAY_ABORT` is not defined.

Definition at line 192 of file `err_hnd.h`.

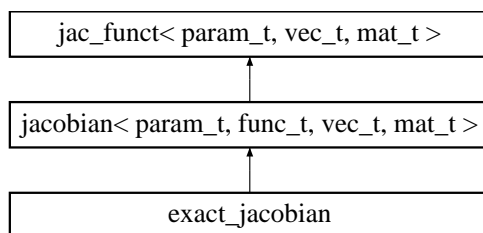
The documentation for this class was generated from the following file:

- [err\\_hnd.h](#)

## 3.62 exact\_jacobian Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for `exact_jacobian`:



### 3.62.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view> class exact_jacobian<
param_t, func_t, vec_t, mat_t >
```

A direct calculation of the `jacobian` using a `deriv` object.

Note that it is sometimes wasteful to use this Jacobian in a root-finding routine and using more approximate Jacobians is more efficient.

Definition at line 296 of file `jacobian.h`.

## Public Member Functions

- [exact\\_jacobian](#) ()
- [int set\\_deriv](#) ([deriv](#)< [ej\\_parms](#), [funct](#)< [ej\\_parms](#) > > &de)  
*Set the derivative object.*
- [virtual int operator\(\)](#) ([size\\_t](#) nv, [vec\\_t](#) &x, [vec\\_t](#) &y, [mat\\_t](#) &jac, [param\\_t](#) &pa)  
*The operator().*

## Data Fields

- [gsl\\_deriv](#)< [ej\\_parms](#), [funct](#)< [ej\\_parms](#) > > [def\\_deriv](#)  
*The default derivative object.*

## Protected Member Functions

- [int dfn](#) ([double](#) x, [double](#) &y, [ej\\_parms](#) &ejp)  
*Desc.*

## Protected Attributes

- [deriv](#)< [ej\\_parms](#), [funct](#)< [ej\\_parms](#) > > \* [dptr](#)  
*Desc.*

## Data Structures

- [struct ej\\_parms](#)  
*Desc.*

The documentation for this class was generated from the following file:

- [jacobian.h](#)

## 3.63 exact\_jacobian::ej\_parms Struct Reference

```
#include <jacobian.h>
```

### 3.63.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view> struct exact_jacobian<
param_t, func_t, vec_t, mat_t >::ej_parms
```

*Desc.*

Definition at line 307 of file [jacobian.h](#).

## Data Fields

- [size\\_t](#) [nv](#)
- [size\\_t](#) [xj](#)
- [size\\_t](#) [yi](#)
- [vec\\_t](#) \* [x](#)
- [vec\\_t](#) \* [y](#)

- param\_t \* pa

The documentation for this struct was generated from the following file:

- jacobian.h

## 3.64 file\_detect Class Reference

```
#include <file_detect.h>
```

### 3.64.1 Detailed Description

Read a (possibly compressed) file and automatically detect the file format.

Really nasty hack. This works by copying the file to a temporary file in /tmp and then uncompressing it using a call to `system("gunzip /tmp/filename")`. When the file is closed, the temporary file is removed using `'rm -f'`.

If the filename ends with ".gz" or ".bz2", then input\_detect will try to uncompress it (using gunzip or bunzip2), otherwise, the file will be treated as normal.

Note that there must be enough disk space in the temporary directory for the uncompressed file or the read will fail.

#### Idea for future

Allow the user to specify the compression commands in configure, or at least specify the path to gzip, bzip2, etc.

Definition at line 57 of file file\_detect.h.

### Public Member Functions

- virtual ~file\_detect ()
- in\_file\_format \* open (const char \*s)  
*Open an input file with the given name.*
- virtual int close ()  
*Close an input file.*
- virtual bool is\_compressed ()  
*Return true if the opened file was originally compressed.*
- virtual bool is\_binary ()  
*Return true if the opened file was a binary file.*

### Protected Attributes

- std::string temp\_filename  
*The temporary filename.*
- std::string user\_filename  
*The user-supplied filename.*
- in\_file\_format \* iffp  
*The input file.*
- bool compressed  
*True if the file was compressed.*
- bool binary  
*True if the file was a binary file.*

### 3.64.2 Member Function Documentation

#### 3.64.2.1 in\_file\_format\* open (const char \* s)

Open an input file with the given name.

If the filename ends with ".gz" or ".bz2", then the file is assumed to be compressed.

It is important to note that the file is not closed until `file_detect::close()` method is called.

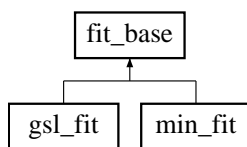
The documentation for this class was generated from the following file:

- file\_detect.h

## 3.65 fit\_base Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for fit\_base::



### 3.65.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view> class fit_base< param_t,
func_t, vec_t, mat_t >
```

Non-linear least-squares fitting base class.

Definition at line 273 of file fit\_base.h.

#### Public Member Functions

- `fit_base()`
- virtual `~fit_base()`
- virtual int `print_iter` (size\_t nv, vec\_t &x, double y, int iter, double value=0.0, double limit=0.0)  
Print out iteration information.
- virtual int `fit` (size\_t ndat, vec\_t &xdat, vec\_t &ydat, vec\_t &yerr, size\_t npar, vec\_t &par, mat\_t &covar, double &chi2, param\_t &pa, func\_t &fitfun)  
Fit the data specified in (xdat,ydat) to the function fitfun with the parameters in par.
- virtual const char \* `type` ()  
Return string denoting type ("fit\_base").

#### Data Fields

- int `verbose`  
An integer describing the verbosity of the output.
- size\_t `n_dat`  
The number of data points.
- size\_t `n_par`  
The number of parameters.

### 3.65.2 Member Function Documentation

**3.65.2.1 virtual int print\_iter (size\_t *ny*, vec\_t & *x*, double *y*, int *iter*, double *value* = 0.0, double *limit* = 0.0)** [inline, virtual]

Print out iteration information.

Depending on the value of the variable `verbose`, this prints out the iteration information. If `verbose=0`, then no information is printed, while if `verbose>1`, then after each iteration, the present values of `x` and `y` are output to `std::cout` along with the iteration number. If `verbose>=2` then each iteration waits for a character.

Definition at line 292 of file `fit_base.h`.

**3.65.2.2 virtual int fit (size\_t *ndat*, vec\_t & *xdat*, vec\_t & *ydat*, vec\_t & *yerr*, size\_t *npar*, vec\_t & *par*, mat\_t & *covar*, double & *chi2*, param\_t & *pa*, func\_t & *fitfun*)** [inline, virtual]

Fit the data specified in (`xdat`,`ydat`) to the function `fitfun` with the parameters in `par`.

The covariance matrix for the parameters is returned in `covar` and the value of  $\chi^2$  is returned in `chi2`.

Reimplemented in `fit_fix_pars`, `gsl_fit`, and `min_fit`.

Definition at line 321 of file `fit_base.h`.

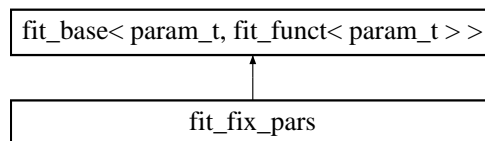
The documentation for this class was generated from the following file:

- `fit_base.h`

## 3.66 fit\_fix\_pars Class Template Reference

```
#include <fit_fix.h>
```

Inheritance diagram for `fit_fix_pars`:



### 3.66.1 Detailed Description

**template<class param\_t, class bool\_vec\_t> class fit\_fix\_pars< param\_t, bool\_vec\_t >**

Multidimensional fitting fixing some parameters and varying others.

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

#### Public Member Functions

- `fit_fix_pars ()`  
*Specify the member function pointer.*
- `virtual ~fit_fix_pars ()`
- `virtual int fit (size_t ndat, ovector_view &xdat, ovector_view &ydat, ovector_view &yerr, size_t npar, ovector_view &par, omatrix_view &covar, double &chi2, param_t &pa, fit_func< param_t > &fitfun)`  
*Fit the data specified in (`xdat`,`ydat`) to the function `fitfun` with the parameters in `par`.*



- virtual int `fit_fix` (size\_t ndat, `ovector_view` &xdat, `ovector_view` &ydat, `ovector_view` &yerr, size\_t npar, `ovector_view` &par, `bool_vec_t` &fix, `omatrix_view` &covar, double &chi2, param\_t &pa, `fit_func_t`< param\_t > &fitfun)  
*Fit function `func` while fixing some parameters as specified in `fix`.*
- int `set_fit` (`fit_base`< param\_t, `fit_func_t`< `fit_fix_pars`, param\_t > > &fitter)  
*Change the base minimizer.*

### Data Fields

- `gsl_fit`< param\_t, `fit_func_t`< `fit_fix_pars`, param\_t > > `def_fit`  
*The default base minimizer.*

### Protected Member Functions

- virtual int `fit_func` (size\_t nv, `ovector_view` &x, double xx, double &y, param\_t &pa)  
*The new function to send to the minimizer.*

### Protected Attributes

- `fit_base`< param\_t, `fit_func_t`< `fit_fix_pars`, param\_t > > \* `fitp`  
*The minimizer.*
- `fit_func_t`< param\_t > \* `funcp`  
*The user-specified function.*
- size\_t `unv`  
*The user-specified number of variables.*
- size\_t `nv_new`  
*The new number of variables.*
- `bool_vec_t` \* `fixp`  
*Specify which parameters to fix.*
- `ovector_view` \* `xp`  
*The user-specified initial vector.*

### Private Member Functions

- `fit_fix_pars` (const `fit_fix_pars` &)
- `fit_fix_pars` & `operator=` (const `fit_fix_pars` &)

## 3.66.2 Member Function Documentation

**3.66.2.1** virtual int `fit` (size\_t ndat, `ovector_view` &xdat, `ovector_view` &ydat, `ovector_view` &yerr, size\_t npar, `ovector_view` &par, `omatrix_view` &covar, double &chi2, param\_t &pa, `fit_func_t`< param\_t > &fitfun) [inline, virtual]

Fit the data specified in (xdat,ydat) to the function `fitfun` with the parameters in `par`.

The covariance matrix for the parameters is returned in `covar` and the value of  $\chi^2$  is returned in `chi2`.

Reimplemented from `fit_base`< param\_t, `fit_func_t`< param\_t > >.

Definition at line 56 of file `fit_fix.h`.

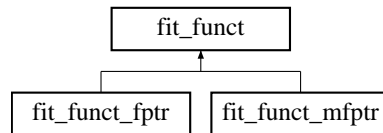
The documentation for this class was generated from the following file:

- `fit_fix.h`

## 3.67 `fit_func` Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for `fit_func`::



### 3.67.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view> class fit_func< param_t, vec_t >
```

Fitting function base.

Definition at line 38 of file `fit_base.h`.

#### Public Member Functions

- `fit_func()`
- `virtual ~fit_func()`
- `virtual int operator()` (`size_t np`, `vec_t &p`, `double x`, `double &y`, `param_t &pa`)  
*Using parameters in `p`, predict `y` given `x`.*

#### Private Member Functions

- `fit_func` (`const fit_func &`)
- `fit_func &operator=` (`const fit_func &`)

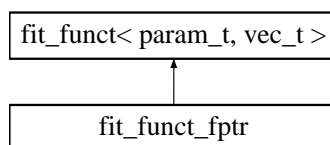
The documentation for this class was generated from the following file:

- `fit_base.h`

## 3.68 `fit_func_fptr` Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for `fit_func_fptr`::



### 3.68.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view> class fit_func_fptr< param_t, vec_t >
```

Function pointer fitting function.

Definition at line 64 of file fit\_base.h.

### Public Member Functions

- [fit\\_funcnt\\_fptr](#) (int(\*fp)(size\_t np, vec\_t &p, double x, double &y, param\_t &pa))  
*Specify a fitting function by a function pointer.*
- virtual [~fit\\_funcnt\\_fptr](#) ()
- virtual int [operator\(\)](#) (size\_t np, vec\_t &p, double x, double &y, param\_t &pa)  
*Using parameters in p, predict y given x.*

### Protected Member Functions

- [fit\\_funcnt\\_fptr](#) ()

### Protected Attributes

- int(\* [fptr](#)) (size\_t np, vec\_t &p, double x, double &y, param\_t &pa)  
*Storage for the user-specified function pointer.*

### Private Member Functions

- [fit\\_funcnt\\_fptr](#) (const [fit\\_funcnt\\_fptr](#) &)
- [fit\\_funcnt\\_fptr](#) & [operator=](#) (const [fit\\_funcnt\\_fptr](#) &)

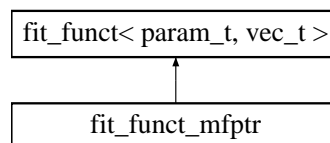
The documentation for this class was generated from the following file:

- fit\_base.h

## 3.69 fit\_funcnt\_mfptr Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for fit\_funcnt\_mfptr::



### 3.69.1 Detailed Description

**template<class tclass, class param\_t, class vec\_t = ovector\_view> class fit\_funcnt\_mfptr< tclass, param\_t, vec\_t >**

Member function pointer fitting function.

Definition at line 107 of file fit\_base.h.

## Public Member Functions

- `fit_func_t_mfp` (tclass \*tp, int(tclass::\*fp)(size\_t np, vec\_t &p, double x, double &y, param\_t &pa))  
*Specify the member function pointer.*
- virtual `~fit_func_t_mfp` ()
- virtual int `operator()` (size\_t np, vec\_t &p, double x, double &y, param\_t &pa)  
*Using parameters in p, predict y given x.*

## Protected Attributes

- int(tclass::\* `fptr`) (size\_t np, vec\_t &p, double x, double &y, param\_t &pa)  
*Storage for the user-specified function pointer.*
- tclass \* `tptr`  
*Storage for the class pointer.*

## Private Member Functions

- `fit_func_t_mfp` (const `fit_func_t_mfp` &)
- `fit_func_t_mfp` & `operator=` (const `fit_func_t_mfp` &)

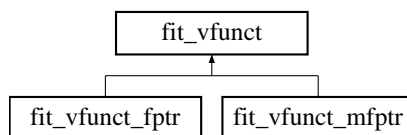
The documentation for this class was generated from the following file:

- fit\_base.h

## 3.70 fit\_vfunct Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for fit\_vfunct::



### 3.70.1 Detailed Description

```
template<class param_t, size_t nvar> class fit_vfunct< param_t, nvar >
```

Fitting function base.

Definition at line 154 of file fit\_base.h.

## Public Member Functions

- `fit_vfunct` ()
- virtual `~fit_vfunct` ()
- virtual int `operator()` (size\_t np, double p[ ], double x, double &y, param\_t &pa)  
*Using parameters in p, predict y given x.*

## Private Member Functions

- **fit\_vfunct** (const **fit\_vfunct** &)
- **fit\_vfunct** & **operator=** (const **fit\_vfunct** &)

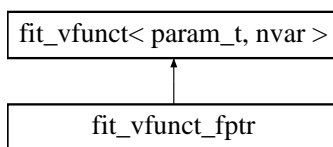
The documentation for this class was generated from the following file:

- fit\_base.h

## 3.71 fit\_vfunct\_fptr Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for fit\_vfunct\_fptr::



### 3.71.1 Detailed Description

```
template<class param_t, size_t nvar> class fit_vfunct_fptr< param_t, nvar >
```

Function pointer fitting function.

Definition at line 180 of file fit\_base.h.

## Public Member Functions

- **fit\_vfunct\_fptr** (int(\*fp)(size\_t np, double p[ ], double x, double &y, param\_t &pa))  
*Specify a fitting function by a function pointer.*
- virtual **~fit\_vfunct\_fptr** ()
- virtual int **operator()** (size\_t np, double p[ ], double x, double &y, param\_t &pa)  
*Using parameters in p, predict y given x.*

## Protected Member Functions

- **fit\_vfunct\_fptr** ()

## Protected Attributes

- int(\* **fptr**) (size\_t np, double p[ ], double x, double &y, param\_t &pa)  
*Storage for the user-specified function pointer.*

## Private Member Functions

- **fit\_vfunct\_fptr** (const **fit\_vfunct\_fptr** &)
- **fit\_vfunct\_fptr** & **operator=** (const **fit\_vfunct\_fptr** &)

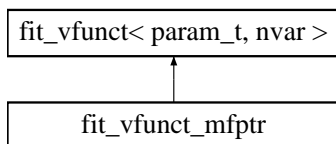
The documentation for this class was generated from the following file:

- fit\_base.h

## 3.72 fit\_vfunct\_mfptr Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for fit\_vfunct\_mfptr::



### 3.72.1 Detailed Description

**template<class tclass, class param\_t, size\_t nvar> class fit\_vfunct\_mfptr< tclass, param\_t, nvar >**

Member function pointer fitting function.

Definition at line 223 of file fit\_base.h.

#### Public Member Functions

- [fit\\_vfunct\\_mfptr](#) (tclass \*tp, int(tclass::\*fp)(size\_t np, double p[ ], double x, double &y, param\_t &pa))  
*Specify the member function pointer.*
- virtual [~fit\\_vfunct\\_mfptr](#) ()
- virtual int [operator\(\)](#) (size\_t np, double p[ ], double x, double &y, param\_t &pa)  
*Using parameters in p, predict y given x.*

#### Protected Attributes

- int(tclass::\* [fptr](#))(size\_t np, double p[ ], double x, double &y, param\_t &pa)  
*Storage for the user-specified function pointer.*
- tclass \* [tptr](#)  
*Storage for the class pointer.*

#### Private Member Functions

- [fit\\_vfunct\\_mfptr](#) (const [fit\\_vfunct\\_mfptr](#) &)
- [fit\\_vfunct\\_mfptr](#) & [operator=](#) (const [fit\\_vfunct\\_mfptr](#) &)

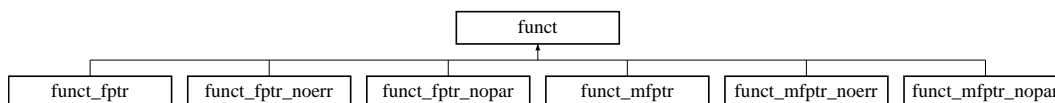
The documentation for this class was generated from the following file:

- fit\_base.h

## 3.73 funct Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for funct::



### 3.73.1 Detailed Description

**template<class param\_t> class funct< param\_t >**

One-dimensional function base.

This class generalizes a function  $y(x)$ .

Definition at line 38 of file `funct.h`.

#### Public Member Functions

- `funct()`
- virtual `~funct()`
- virtual int `operator()` (double x, double &y, param\_t &pa)  
*The overloaded operator().*
- virtual double `operator()` (double x, param\_t &pa)  
*The overloaded operator().*

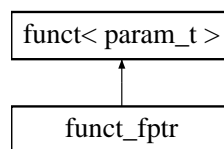
The documentation for this class was generated from the following file:

- `funct.h`

## 3.74 `funct_fptr` Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for `funct_fptr`:



### 3.74.1 Detailed Description

**template<class param\_t> class funct\_fptr< param\_t >**

Function pointer to a function.

Definition at line 76 of file `funct.h`.

#### Public Member Functions

- `funct_fptr` (int(\*fp)(double x, double &y, param\_t &pa))  
*Specify the function pointer.*
- virtual `~funct_fptr()`
- virtual int `operator()` (double x, double &y, param\_t &pa)  
*The overloaded operator().*
- virtual double `operator()` (double x, param\_t &pa)  
*The overloaded operator().*

### Protected Member Functions

- `funct_fptr()`

### Protected Attributes

- `int(* fptr)(double x, double &y, param_t &pa)`  
*Storage for the function pointer.*

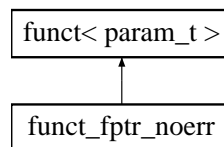
The documentation for this class was generated from the following file:

- `funct.h`

## 3.75 `funct_fptr_noerr` Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for `funct_fptr_noerr`:



### 3.75.1 Detailed Description

```
template<class param_t> class funct_fptr_noerr< param_t >
```

Function pointer to a function.

Definition at line 125 of file `funct.h`.

### Public Member Functions

- `funct_fptr_noerr` (`double(*fp)(double x, param_t &pa)`)  
*Specify the function pointer.*
- `virtual ~funct_fptr_noerr()`
- `virtual int operator()` (`double x, double &y, param_t &pa`)  
*The overloaded operator().*
- `virtual double operator()` (`double x, param_t &pa`)  
*The overloaded operator().*

### Protected Attributes

- `double(* fptr)(double x, param_t &pa)`  
*Storage for the function pointer.*

The documentation for this class was generated from the following file:

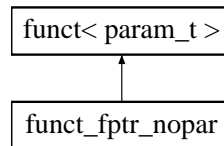
- `funct.h`



## 3.76 `funct_fptr_nopar` Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for `funct_fptr_nopar`:



### 3.76.1 Detailed Description

```
template<class param_t> class funct_fptr_nopar< param_t >
```

Function pointer to a function.

Definition at line 171 of file `funct.h`.

#### Public Member Functions

- `funct_fptr_nopar` (`double(*fp)(double x)`)  
*Specify the function pointer.*
- virtual `~funct_fptr_nopar` ()
- virtual int `operator()` (`double x`, `double &y`, `param_t &pa`)  
*The overloaded operator().*
- virtual double `operator()` (`double x`, `param_t &pa`)  
*The overloaded operator().*

#### Protected Attributes

- `double(* fptr)` (`double x`)  
*Storage for the function pointer.*

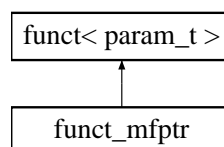
The documentation for this class was generated from the following file:

- `funct.h`

## 3.77 `funct_mfptr` Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for `funct_mfptr`:



### 3.77.1 Detailed Description

**template<class tclass, class param\_t> class funct\_mfptr< tclass, param\_t >**

Member function pointer to a one-dimensional function.

Definition at line 217 of file funct.h.

#### Public Member Functions

- [funct\\_mfptr](#) (tclass \*tp, int(tclass::\*fp)(double x, double &y, param\_t &pa))  
*Specify the member function pointer.*
- virtual [~funct\\_mfptr](#) ()
- virtual int [operator\(\)](#) (double x, double &y, param\_t &pa)  
*The overloaded operator().*
- virtual double [operator\(\)](#) (double x, param\_t &pa)  
*The overloaded operator().*

#### Protected Attributes

- int(tclass::\* [fptr](#) )(double x, double &y, param\_t &pa)  
*Storage for the member function pointer.*
- tclass \* [tptr](#)  
*Store the pointer to the class instance.*

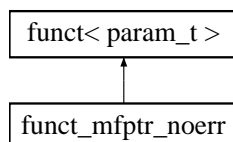
The documentation for this class was generated from the following file:

- funct.h

## 3.78 funct\_mfptr\_noerr Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for funct\_mfptr\_noerr::



### 3.78.1 Detailed Description

**template<class tclass, class param\_t> class funct\_mfptr\_noerr< tclass, param\_t >**

Member function pointer to a one-dimensional function.

Definition at line 267 of file funct.h.

#### Public Member Functions

- [funct\\_mfptr\\_noerr](#) (tclass \*tp, double(tclass::\*fp)(double x, param\_t &pa))  
*Specify the member function pointer.*

- virtual `~funct_mfptr_noerr()`
- virtual int `operator()` (double x, double &y, param\_t &pa)  
*The overloaded operator().*
- virtual double `operator()` (double x, param\_t &pa)  
*The overloaded operator().*

### Protected Attributes

- double(tclass::\* `fptr`)(double x, param\_t &pa)  
*Storage for the member function pointer.*
- tclass \* `tptr`  
*Store the pointer to the class instance.*

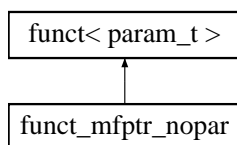
The documentation for this class was generated from the following file:

- funct.h

## 3.79 funct\_mfptr\_nopar Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for funct\_mfptr\_nopar::



### 3.79.1 Detailed Description

```
template<class tclass, class param_t> class funct_mfptr_nopar< tclass, param_t >
```

Member function pointer to a one-dimensional function.

Definition at line 318 of file funct.h.

### Public Member Functions

- `funct_mfptr_nopar` (tclass \*tp, double(tclass::\*fp)(double x))  
*Specify the member function pointer.*
- virtual `~funct_mfptr_nopar()`
- virtual int `operator()` (double x, double &y, param\_t &pa)  
*The overloaded operator().*
- virtual double `operator()` (double x, param\_t &pa)  
*The overloaded operator().*

### Protected Attributes

- double(tclass::\* `fptr`)(double x)  
*Storage for the member function pointer.*
- tclass \* `tptr`  
*Store the pointer to the class instance.*

The documentation for this class was generated from the following file:

- funct.h

## 3.80 gaussian\_2d Class Template Reference

```
#include <gaussian_2d.h>
```

### 3.80.1 Detailed Description

```
template<class rng_t> class gaussian_2d< rng_t >
```

Generate two random numbers from a normal distribution.

#### Todo

Double check that sigma is implemented correctly

Definition at line 38 of file gaussian\_2d.h.

### Public Member Functions

- void [random](#) (double sigma, double &x, double &y)  
*Generate two numbers from a distribution with zero mean and standard deviation sigma.*

### Data Fields

- rng\_t [r](#)

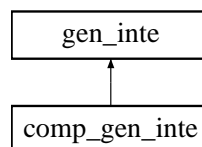
The documentation for this class was generated from the following file:

- gaussian\_2d.h

## 3.81 gen\_inte Class Template Reference

```
#include <gen_inte.h>
```

Inheritance diagram for gen\_inte::



### 3.81.1 Detailed Description

```
template<class param_t, class func_t, class lfunc_t, class ufunc_t, class vec_t = ovector_view> class gen_inte< param_t,
func_t, lfunc_t, ufunc_t, vec_t >
```

Generalized multi-dimensional integration base.

In order to allow the user to specify only three functions (for the integrand, the lower limits, and the upper limits) the first integer variable is used to distinguish among the variable limits. So the function  $a_0()$  is just `lower(0,NULL,vp)` where `vp` is a void pointer, the function  $a_1$  is `lower(1,x,vp)` where `x` is a 1-dimensional vector, and the function  $a_i$  is `lower(i,x,vp)` where `x` is an  $i$ -dimensional vector. Similarly, the function  $b_i$  is `upper(i,x,vp)` where `x` is an  $i$ -dimensional vector.

Definition at line 44 of file `gen_inte.h`.

### Public Member Functions

- `gen_inte()`
- `double ginteg(func_t &func, size_t ndim, lfunc_t &a, ufunc_t &b, param_t &pa)`  
*Integrate function `func` from  $x_i = f_i(x_i)$  to  $x_i = g_i(x_i)$  for  $0 < i < \text{ndim} - 1$ .*
- `int ginteg_err(func_t &func, size_t ndim, lfunc_t &a, ufunc_t &b, param_t &pa, double &res, double &err)`  
*Integrate function `func` from  $x_i = f_i(x_i)$  to  $x_i = g_i(x_i)$  for  $0 < i < \text{ndim} - 1$ .*
- `double get_error()`  
*Return the error in the result from the last call to `ginteg()` or `ginteg_err()`.*
- `const char * type()`  
*Return string denoting type ("`gen_inte`").*

### Data Fields

- `int verbose`  
*Verbosity.*
- `double tolf`  
*The maximum "uncertainty" in the value of the integral.*

### Protected Attributes

- `double interror`  
*The uncertainty for the last integration computation.*

## 3.81.2 Member Function Documentation

### 3.81.2.1 double get\_error() [inline]

Return the error in the result from the last call to `ginteg()` or `ginteg_err()`.

This will quietly return zero if no integrations have been performed.

Definition at line 92 of file `gen_inte.h`.

The documentation for this class was generated from the following file:

- `gen_inte.h`

## 3.82 gen\_test\_number Class Template Reference

```
#include <misc.h>
```

### 3.82.1 Detailed Description

```
template<size_t tot> class gen_test_number< tot >
```

Generate number sequence for testing.

A class which generates `tot` numbers from -1 to 1, making sure to include -1, 1, 0, and numbers near -1, 0 and 1 (so long as `tot` is sufficiently large). If `gen()` is called more than `tot` times, it just recycles through the list again. The template argument `tot` should probably be greater than or equal to three.

At present, this is used to generate combinations of pathological coefficients for testing the polynomial solvers.

Definition at line 169 of file `misc.h`.

### Public Member Functions

- `gen_test_number()`
- `double gen()`  
*Return the next number in the sequence.*

### Protected Attributes

- `int n`  
*Count number of numbers generated.*
- `double fact`  
*A constant factor for the argument to `tanh()`.*

The documentation for this class was generated from the following file:

- `misc.h`

## 3.83 gm\_parms Struct Template Reference

```
#include <gsl_vegas.h>
```

### 3.83.1 Detailed Description

```
template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_view> struct gm_parms< param_t,
func_t, rng_t, vec_t >
```

The parameters for the GSL function pointer.

Definition at line 92 of file `gsl_vegas.h`.

### Data Fields

- `func_t * func`
- `void * vp`
- `func_t * func`
- `void * vp`
- `func_t * func`
- `void * vp`

The documentation for this struct was generated from the following files:

- `gsl_vegas.h`
- `gsl_miser.h`
- `gsl_monte.h`

## 3.84 gm\_parms Struct Template Reference

```
#include <gsl_vegas.h>
```

### 3.84.1 Detailed Description

```
template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_view> struct gm_parms< param_t, func_t, rng_t, vec_t >
```

The parameters for the GSL function pointer.

Definition at line 92 of file `gsl_vegas.h`.

#### Data Fields

- `func_t * func`
- `void * vp`
- `func_t * func`
- `void * vp`
- `func_t * func`
- `void * vp`

The documentation for this struct was generated from the following files:

- `gsl_vegas.h`
- `gsl_miser.h`
- `gsl_monte.h`

## 3.85 gm\_parms Struct Template Reference

```
#include <gsl_vegas.h>
```

### 3.85.1 Detailed Description

```
template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_view> struct gm_parms< param_t, func_t, rng_t, vec_t >
```

The parameters for the GSL function pointer.

Definition at line 92 of file `gsl_vegas.h`.

#### Data Fields

- `func_t * func`
- `void * vp`
- `func_t * func`
- `void * vp`
- `func_t * func`
- `void * vp`

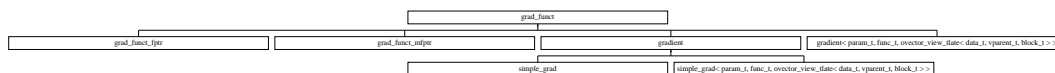
The documentation for this struct was generated from the following files:

- `gsl_vegas.h`
  - `gsl_miser.h`
  - `gsl_monte.h`
-

## 3.86 grad\_func Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for grad\_func::



### 3.86.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view> class grad_func< param_t, vec_t >
```

Base class for a [gradient](#) function.

Definition at line 36 of file multi\_min.h.

#### Public Member Functions

- virtual [~grad\\_func](#) ()
- virtual int [operator\(\)](#) (size\_t nv, vec\_t &x, vec\_t &g, param\_t &pa)  
Compute the [gradient](#)  $g$  at the point  $x$ .

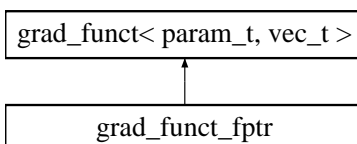
The documentation for this class was generated from the following file:

- multi\_min.h

## 3.87 grad\_func\_ptr Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for grad\_func\_ptr::



### 3.87.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view> class grad_func_ptr< param_t, vec_t >
```

Function pointer to a [gradient](#).

Definition at line 52 of file multi\_min.h.

#### Public Member Functions

- [grad\\_func\\_ptr](#) (int(\*fp)(size\_t nv, vec\_t &x, vec\_t &g, param\_t &pa))  
Specify the function pointer.
- virtual [~grad\\_func\\_ptr](#) ()
- virtual int [operator\(\)](#) (size\_t nv, vec\_t &x, vec\_t &g, param\_t &pa)  
Compute the [gradient](#)  $g$  at the point  $x$ .



### Protected Attributes

- `int(* fptr)(size_t nv, vec_t &x, vec_t &g, param_t &pa)`  
*The function pointer.*

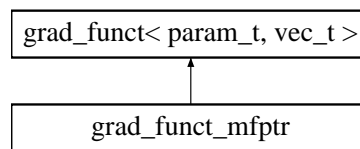
The documentation for this class was generated from the following file:

- `multi_min.h`

## 3.88 grad\_funct\_mfptr Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for `grad_funct_mfptr`:



### 3.88.1 Detailed Description

**template<class tclass, class param\_t, class vec\_t = ovector\_view> class grad\_funct\_mfptr< tclass, param\_t, vec\_t >**

Member function pointer to a [gradient](#).

Definition at line 93 of file `multi_min.h`.

### Public Member Functions

- `grad\_funct\_mfptr(tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &g, param_t &pa))`  
*Specify the member function pointer.*
- `virtual ~grad\_funct\_mfptr()`
- `virtual int operator()(size_t nv, vec_t &x, vec_t &g, param_t &pa)`  
*Compute the [gradient](#) g at the point x.*

### Protected Attributes

- `int(tclass::* fptr)(size_t nv, vec_t &x, vec_t &g, param_t &pa)`  
*Member function pointer.*
- `tclass * tptr`  
*Class pointer.*

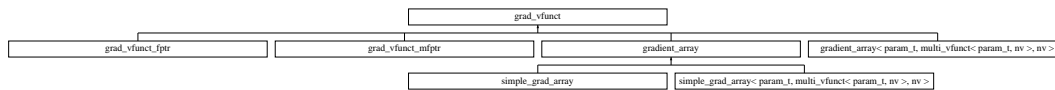
The documentation for this class was generated from the following file:

- `multi_min.h`

## 3.89 grad\_vfunct Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for grad\_vfunct::



### 3.89.1 Detailed Description

```
template<class param_t, size_t nv> class grad_vfunct< param_t, nv >
```

Base class for a [gradient](#) function using arrays.

Definition at line 212 of file multi\_min.h.

#### Public Member Functions

- virtual [~grad\\_vfunct](#) ()
- virtual int [operator\(\)](#) (size\_t nvar, double x[nv], double g[nv], param\_t &pa)  
*Compute the [gradient](#)  $g$  at the point  $x$ .*

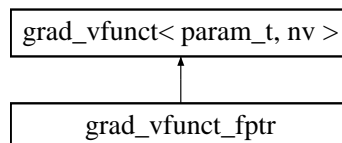
The documentation for this class was generated from the following file:

- multi\_min.h

## 3.90 grad\_vfunct\_fptr Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for grad\_vfunct\_fptr::



### 3.90.1 Detailed Description

```
template<class param_t, size_t nv> class grad_vfunct_fptr< param_t, nv >
```

Function pointer to a [gradient](#).

Definition at line 230 of file multi\_min.h.

#### Public Member Functions

- [grad\\_vfunct\\_fptr](#) (int(\*fp)(size\_t nv, double x[nv], double g[nv], param\_t &pa))  
*Specify the member function pointer.*

- virtual [~grad\\_vfunct\\_fptr](#) ()
- virtual int [operator\(\)](#) (size\_t nvar, double x[nv], double g[nv], param\_t &pa)  
*Compute the [gradient](#)  $\mathfrak{g}$  at the point  $\mathfrak{x}$ .*

### Protected Attributes

- int(\* [fptr](#)) (size\_t nvar, double x[nv], double g[nv], param\_t &pa)  
*Function pointer.*

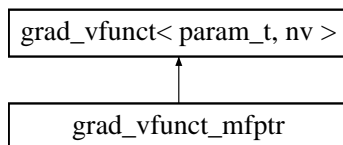
The documentation for this class was generated from the following file:

- multi\_min.h

## 3.91 grad\_vfunct\_mfptr Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for grad\_vfunct\_mfptr::



### 3.91.1 Detailed Description

```
template<class tclass, class param_t, size_t nv> class grad_vfunct_mfptr< tclass, param_t, nv >
```

Member function pointer to a [gradient](#).

Definition at line 273 of file multi\_min.h.

### Public Member Functions

- [grad\\_vfunct\\_mfptr](#) (tclass \*tp, int(tclass::\*fp)(size\_t nvar, double x[nv], double g[nv], param\_t &pa))  
*Specify the member function pointer.*
- virtual [~grad\\_vfunct\\_mfptr](#) ()
- virtual int [operator\(\)](#) (size\_t nvar, double x[nv], double g[nv], param\_t &pa)  
*Compute the [gradient](#)  $\mathfrak{g}$  at the point  $\mathfrak{x}$ .*

### Protected Attributes

- int(tclass::\* [fptr](#)) (size\_t nvar, double x[nv], double g[nv], param\_t &pa)  
*Member function pointer.*
- tclass \* [tptr](#)  
*Class pointer.*

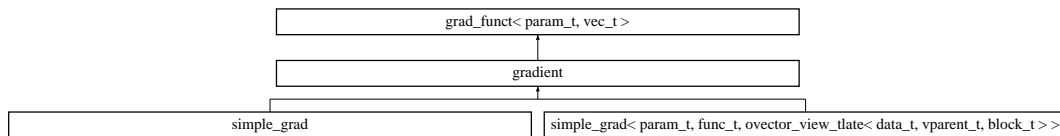
The documentation for this class was generated from the following file:

- multi\_min.h

## 3.92 gradient Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for gradient::



### 3.92.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t = ovector\_view> class gradient< param\_t, func\_t, vec\_t >**

Base class for automatically computing gradients.

Definition at line 138 of file multi\_min.h.

#### Public Member Functions

- virtual [~gradient](#) ()
- virtual int [set\\_function](#) (func\_t &f)  
Set the function to compute the [gradient](#) of.
- virtual int [operator\(\)](#) (size\_t nv, vec\_t &x, vec\_t &g, param\_t &pa)  
Compute the [gradient](#)  $\mathbf{g}$  at the point  $\mathbf{x}$ .

#### Protected Attributes

- func\_t \* [func](#)  
A pointer to the user-specified function.

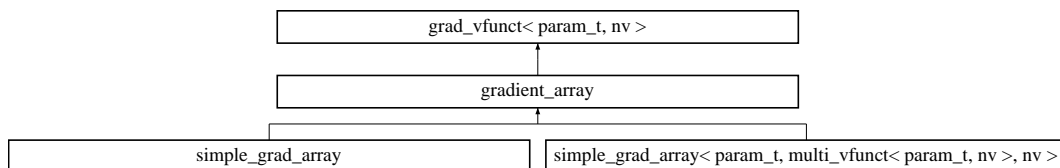
The documentation for this class was generated from the following file:

- multi\_min.h

## 3.93 gradient\_array Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for gradient\_array::



### 3.93.1 Detailed Description

**template<class param\_t, class func\_t, size\_t nv> class gradient\_array< param\_t, func\_t, nv >**

Base class for automatically computing gradients with arrays.

Definition at line 319 of file multi\_min.h.

#### Public Member Functions

- virtual [~gradient\\_array](#) ()
- virtual int [set\\_function](#) (func\_t &f)  
Set the function to compute the *gradient* of.
- virtual int [operator\(\)](#) (size\_t nvar, double x[nv], double g[nv], param\_t &pa)  
Compute the *gradient*  $g$  at the point  $x$ .

#### Protected Attributes

- func\_t \* [func](#)  
A pointer to the user-specified function.

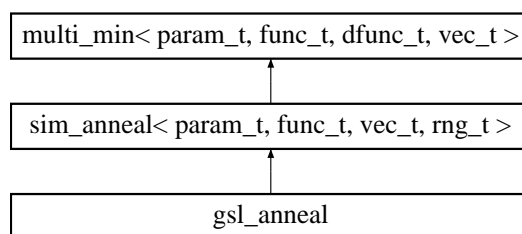
The documentation for this class was generated from the following file:

- multi\_min.h

## 3.94 gsl\_anneal Class Template Reference

```
#include <gsl_anneal.h>
```

Inheritance diagram for gsl\_anneal::



### 3.94.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t = ovector\_view, class rng\_t = gsl\_rnga> class gsl\_anneal< param\_t, func\_t, vec\_t, rng\_t >**

Multidimensional minimization by simulated annealing (GSL).

Minimize by simulated annealing for an arbitrary temperature schedule and random number generator. A move is defined by

$$x_{i,\text{new}} = \text{stepsize}(2u_i - 1) + x_{i,\text{old}}$$

where the  $u_i$  are random numbers between 0 and 1. The random number generator and temperature schedule are set in the parent, [sim\\_anneal](#). The variables [minimize::tolx](#) and [minimize::tolf](#) are not used

### Todo

Implement different stepsizes for the different dimensions

### Idea for future

Implement a more general routine which would allow the solution of discrete problems like the Traveling Salesman problem.

Definition at line 55 of file gsl\_anneal.h.

### Public Member Functions

- virtual `~gsl_anneal()`
- virtual `int mmin` (size\_t nvar, vec\_t &x0, double &fmin, param\_t &pa, func\_t &func)  
*Calculate the minimum fmin of func w.r.t the array x0 of size nvar.*
- virtual `const char * type` ()  
*Return string denoting type ("gsl\_anneal").*

### Data Fields

- double `step_size`  
*Size of step (default 10.0).*
- double `boltz`  
*Boltzmann factor (default 1.0).*

### Protected Member Functions

- `int allocate` (size\_t n, double usep=10.0, double boltz\_factor=1.0)  
*Allocate memory for a minimizer over n dimensions with stepsize step and Boltzmann factor boltz\_factor.*
- `int free` ()  
*Free allocated memory.*
- `int step` (vec\_t &sx, int nvar)  
*Make a step to a new attempted minimum.*

### Protected Attributes

Storage for present, next, and best vectors

- `ovector x`
- `ovector new_x`
- `ovector best_x`

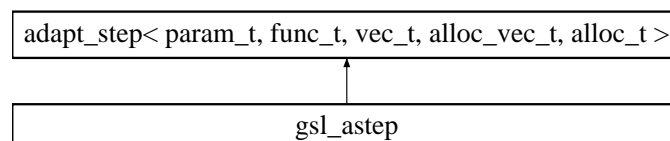
The documentation for this class was generated from the following file:

- `gsl_anneal.h`

## 3.95 gsl\_astep Class Template Reference

```
#include <gsl_astep.h>
```

Inheritance diagram for `gsl_astep`:



### 3.95.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>
class gsl_astep< param_t, func_t, vec_t, alloc_vec_t, alloc_t >
```

Adaptive stepper (GSL).

#### Todo

Finish implementing the scaled "control"

#### Todo

Allow user to find out how many steps were taken, etc.

Definition at line 43 of file `gsl_astep.h`.

### Public Member Functions

- [gsl\\_astep\(\)](#)
- virtual [~gsl\\_astep\(\)](#)
- template<class svec\_t>  
int [set\\_scale](#) (size\_t nscal, const svec\_t &scale)  
*Set scalings.*
- virtual int [astep](#) (double &x, double &h, double xmax, size\_t n, vec\_t &y, param\_t &pa, func\_t &derivs)  
*Make an adaptive integration step of the system derivs.*
- virtual int [astep\\_derivs](#) (double &x, double &h, double xmax, size\_t n, vec\_t &y, vec\_t &dydx, param\_t &pa, func\_t &derivs)  
*Make an adaptive integration step of the system derivs.*

### Data Fields

- [gsl\\_ode\\_control](#) con  
*Control specification.*

### Protected Member Functions

- int [hadjust](#) (size\_t dim, unsigned int ord, const vec\_t &y, vec\_t &yerr, vec\_t &yp, double \*h)  
*Automatically adjust step-size.*
- int [evolve\\_apply](#) ([gsl\\_odeiv\\_evolve](#) \*e, size\_t nvar, param\_t &pa, func\_t &derivs, double \*t, double t1, double \*h, vec\_t &y)  
*Apply the evolution for the next adaptive step.*

### Protected Attributes

- size\_t [ndim](#)  
*Size of allocated vectors.*
- alloc\_t [ao](#)  
*Memory allocator for objects of type alloc\_vec\_t.*
- size\_t [sdim](#)  
*Number of scalings.*
- double \* [scale\\_abs](#)  
*Scalings.*

## Data Structures

- struct [gsl\\_ode\\_control](#)  
*Control structure.*
- class [gsl\\_odeiv\\_evolve](#)  
*The GSL evolution object for [gsl\\_astep](#) [protected subclass].*

### 3.95.2 Member Function Documentation

**3.95.2.1 virtual int astep (double & x, double & h, double xmax, size\_t n, vec\_t & y, param\_t & pa, func\_t & derivs)**  
[inline, virtual]

Make an adaptive integration step of the system `derivs`.

This attempts to take a step of size `h` from the point `x` of an `n`-dimensional system `derivs` starting with `y`. On exit, `x` and `y` contain the new values at the end of the step and `h` contains the size of the step.

Reimplemented from [adapt\\_step](#).

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

**3.95.2.2 virtual int astep\_derivs (double & x, double & h, double xmax, size\_t n, vec\_t & y, vec\_t & dydx, param\_t & pa, func\_t & derivs)** [inline, virtual]

Make an adaptive integration step of the system `derivs`.

This attempts to take a step of size `h` from the point `x` of an `n`-dimensional system `derivs` starting with `y`. On exit, `x` and `y` contain the new values at the end of the step and `h` contains the size of the step.

Reimplemented from [adapt\\_step](#).

Definition at line 336 of file `gsl_astep.h`.

The documentation for this class was generated from the following file:

- `gsl_astep.h`

## 3.96 gsl\_astep::gsl\_ode\_control Struct Reference

```
#include <gsl_astep.h>
```

### 3.96.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>
struct gsl_astep< param_t, func_t, vec_t, alloc_vec_t, alloc_t >::gsl_ode_control
```

Control structure.

Definition at line 236 of file `gsl_astep.h`.

## Data Fields

- double [eps\\_abs](#)  
*Absolute precision.*
- double [eps\\_rel](#)  
*Relative precision.*
- double [a\\_y](#)  
*Function scaling factor.*



- double [a\\_dydt](#)  
*Derivative scaling factor.*
- bool [standard](#)  
*Use standard or scaled algorithm.*

The documentation for this struct was generated from the following file:

- [gsl\\_astep.h](#)

## 3.97 gsl\_astep::gsl\_odeiv\_evolve Class Reference

```
#include <gsl_astep.h>
```

### 3.97.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t = ovector\_view, class alloc\_vec\_t = ovector, class alloc\_t = ovector\_alloc>  
class gsl\_astep< param\_t, func\_t, vec\_t, alloc\_vec\_t, alloc\_t >::gsl\_odeiv\_evolve**

The GSL evolution object for [gsl\\_astep](#) [protected subclass].

Definition at line 66 of file [gsl\\_astep.h](#).

#### Data Fields

- size\_t [dimension](#)  
*Number of differential equations.*
- alloc\_vec\_t [y0](#)  
*The values of the functions.*
- alloc\_vec\_t [yerr](#)  
*The uncertainty.*
- alloc\_vec\_t [dydt\\_in](#)  
*The input derivatives.*
- alloc\_vec\_t [dydt\\_out](#)  
*The output derivatives.*
- size\_t [ndim](#)  
*Size of the memory allocation.*
- double [last\\_step](#)  
*The size of the last step.*
- unsigned long int [count](#)  
*The number of steps (?).*
- unsigned long int [failed\\_steps](#)  
*The number of failed steps.*

The documentation for this class was generated from the following file:

- [gsl\\_astep.h](#)

## 3.98 gsl\_chebapp Class Template Reference

```
#include <gsl_chebapp.h>
```

### 3.98.1 Detailed Description

**template<class param\_t, class func\_t> class gsl\_chebapp< param\_t, func\_t >**

Chebyshev approximation (GSL).

Approximate a function using a Chebyshev series:

$$f(x) = \sum_n c_n T_n(x) \quad \text{where} \quad T_n(x) = \cos(n \arccos x)$$

Definition at line 44 of file `gsl_chebapp.h`.

#### Public Member Functions

- [gsl\\_chebapp](#) ()
- [int init](#) (func\_t &func, double a, double b, param\_t &vp)   
 *Initialize a Chebyshev approximation of the function func over the interval from a to b.*
- [int set\\_order](#) (size\_t o)   
 *Set the order (default 5).*
- [double eval](#) (double x)   
 *Evaluate the approximation.*
- [gsl\\_chebapp \\* deriv](#) ()   
 *Return a pointer to an approximation to the derivative.*
- [gsl\\_chebapp \\* inte](#) ()   
 *Return a pointer to an approximation to the integral.*
- [double get\\_coefficient](#) (size\_t ix)   
 *Get the coefficient.*

### 3.98.2 Member Function Documentation

#### 3.98.2.1 int init (func\_t &func, double a, double b, param\_t &vp) [inline]

Initialize a Chebyshev approximation of the function `func` over the interval from `a` to `b`.

The interval must be specified so that  $a < b$ .

Definition at line 58 of file `gsl_chebapp.h`.

#### 3.98.2.2 int set\_order (size\_t o) [inline]

Set the order (default 5).

The function `init()` must be called after calling `set_order()` to reinitialize the series for the new order.

Definition at line 95 of file `gsl_chebapp.h`.

#### 3.98.2.3 gsl\_chebapp\* deriv () [inline]

Return a pointer to an approximation to the derivative.

The new `gsl_chebapp` object is allocated by `new`, and the memory should be deallocated using `delete` by the user.

Definition at line 120 of file `gsl_chebapp.h`.

#### 3.98.2.4 gsl\_chebapp\* inte () [inline]

Return a pointer to an approximation to the integral.

The new [gsl\\_chebapp](#) object is allocated by new, and the memory should be deallocated using delete by the user.

Definition at line 139 of file gsl\_chebapp.h.

### 3.98.2.5 double get\_coefficient (size\_t ix) [inline]

Get the coefficient.

Legal values of the argument are 0 to order+1

Definition at line 157 of file gsl\_chebapp.h.

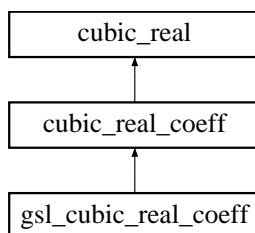
The documentation for this class was generated from the following file:

- gsl\_chebapp.h

## 3.99 gsl\_cubic\_real\_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for gsl\_cubic\_real\_coeff::



### 3.99.1 Detailed Description

Solve a cubic with real coefficients and complex roots (GSL).

Definition at line 460 of file poly.h.

#### Public Member Functions

- virtual `~gsl_cubic_real_coeff()`
- virtual `int solve_rc(const double a3, const double b3, const double c3, const double d3, double &x1, std::complex< double > &x2, std::complex< double > &x3)`  
*Solves the polynomial  $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$  giving the real solution  $x = x_1$  and two complex solutions  $x = x_1, x = x_2$ , and  $x = x_3$ .*
- `const char * type()`  
*Return a string denoting the type ("gsl\_cubic\_real\_coeff").*

#### Protected Member Functions

- `int gsl_poly_complex_solve_cubic2(double a, double b, double c, gsl_complex *z0, gsl_complex *z1, gsl_complex *z2)`  
*An alternative to `gsl_poly_complex_solve_cubic()`.*

### 3.99.2 Member Function Documentation

**3.99.2.1** `int gsl_poly_complex_solve_cubic2(double a, double b, double c, gsl_complex *z0, gsl_complex *z1, gsl_complex *z2)` [protected]

An alternative to `gsl_poly_complex_solve_cubic()`.

This is an alternative to the function `gsl_poly_complex_solve_cubic()` with some small corrections.

### Todo

Demonstrate that this works better than the original GSL version.

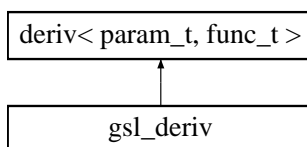
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.100 gsl\_deriv Class Template Reference

```
#include <gsl_deriv.h>
```

Inheritance diagram for `gsl_deriv`:



### 3.100.1 Detailed Description

```
template<class param_t, class func_t> class gsl_deriv< param_t, func_t >
```

Numerical differentiation (GSL).

This class computes the numerical derivative of a function. The stepsize `h` should be specified before use. If similar functions are being differentiated in succession, the user may be able to increase the speed of later derivatives by setting the new stepsize equal to the optimized stepsize from the previous differentiation, by setting `h` to `h_opt`.

#### Note:

Second and third derivatives are computed by naive nested applications of the formula for the first derivative and the uncertainty for these will likely be underestimated.

### Idea for future

Include the forward and backward GSL derivatives

Definition at line 53 of file `gsl_deriv.h`.

### Public Member Functions

- `gsl_deriv()`
- `virtual ~gsl_deriv()`
- `virtual int calc_err(double x, param_t &pa, func_t &func, double &dwdx, double &err)`  
Calculate the first derivative of `func` w.r.t. `x` and uncertainty.
- `virtual const char * type()`  
Return string denoting type ("gsl\_deriv").

## Data Fields

- double **h**  
*Initial stepsize.*
- double **h\_opt**  
*The last value of the optimized stepsize.*

## Protected Member Functions

- virtual int **calc\_err\_int** (double x, typename **deriv**< param\_t, func\_t >::dpars &pa, **funct**< typename **deriv**< param\_t, func\_t >::dpars > &func, double &dfdx, double &err)  
*Internal version of **calc\_err()** for second and third derivatives.*
- template<class func2\_t, class param2\_t>  
int **central\_deriv** (double x, double hh, double &result, double &abserr\_round, double &abserr\_trunc, func2\_t &func, param2\_t &pa)  
*Compute derivative using 5-point rule.*

### 3.100.2 Member Function Documentation

#### 3.100.2.1 int central\_deriv (double x, double hh, double & result, double & abserr\_round, double & abserr\_trunc, func2\_t & func, param2\_t & pa) [inline, protected]

Compute derivative using 5-point rule.

Compute the derivative using the 5-point rule (x-h, x-h/2, x, x+h/2, x+h) and the error using the difference between the 5-point and the 3-point rule (x-h,x,x+h). Note that the central point is not used for either.

This must be a class template because it is used by both **calc\_err()** and **calc\_err\_int()**.

Definition at line 199 of file gsl\_deriv.h.

### 3.100.3 Field Documentation

#### 3.100.3.1 double h

Initial stepsize.

This should be specified before a call to **calc()** or **calc\_err()**. If it is zero, then  $x10^{-4}$  will used, or if x is zero, then  $10^{-4}$  will be used.

Definition at line 70 of file gsl\_deriv.h.

#### 3.100.3.2 double h\_opt

The last value of the optimized stepsize.

This is initialized to zero in the constructor and set by **calc\_err()** to the most recent value of the optimized stepsize.

Definition at line 79 of file gsl\_deriv.h.

The documentation for this class was generated from the following file:

- gsl\_deriv.h

## 3.101 gsl\_fft Class Reference

```
#include <gsl_fft.h>
```

### 3.101.1 Detailed Description

Real mixed-radix fast Fourier transform.

This is a simple wrapper for the GSL FFT functions which automatically allocates the necessary memory.

Definition at line 42 of file `gsl_fft.h`.

#### Public Member Functions

- `int transform` (int n, double \*x)  
*Perform the FFT transform.*
- `int inverse_transform` (int n, double \*x)  
*Perform the inverse FFT transform.*

#### Protected Member Functions

- `int mem_resize` (int new\_size)  
*Reallocate memory.*

#### Protected Attributes

- `int mem_size`  
*The current memory size.*
- `gsl_fft_real_workspace * work`  
*The GSL workspace.*
- `gsl_fft_real_wavetable * real`  
*The [table](#) for the forward transform.*
- `gsl_fft_halfcomplex_wavetable * hc`  
*The [table](#) for the inverse transform.*

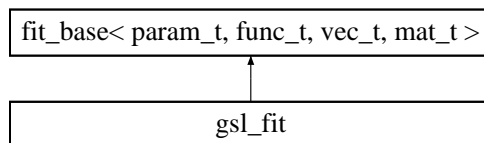
The documentation for this class was generated from the following file:

- `gsl_fft.h`

## 3.102 gsl\_fit Class Template Reference

```
#include <gsl_fit.h>
```

Inheritance diagram for `gsl_fit`:



### 3.102.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view, class bool_vec_t = bool *>
class gsl_fit< param_t, func_t, vec_t, mat_t, bool_vec_t >
```

Non-linear least-squares fitting class (GSL).

The GSL-based fitting class using a Levenberg-Marquardt type algorithm. The algorithm stops when

$$|dx_i| < \text{epsabs} + \text{epsrel} \times |x_i|$$

where  $dx$  is the last step and  $x$  is the current position. If `test_gradient` is true, then additionally `fit()` requires that

$$\sum_i |g_i| < \text{epsabs}$$

where  $g_i$  is the  $i$ -th component of the `gradient` of the function  $\Phi(x)$  where

$$\Phi(x) = ||F(x)||^2$$

### Todo

Properly generalize other vector types than `ovector_view`

### Todo

Allow the user to specify the derivatives

### Todo

Fix so that the user can specify automatic scaling of the fitting parameters, where the initial guess are used for scaling so that the fitting parameters are near unity.

Definition at line 66 of file `gsl_fit.h`.

## Public Member Functions

- `gsl_fit()`
- virtual `~gsl_fit()`
- virtual int `fit` (size\_t ndat, vec\_t &xdat, vec\_t &ydat, vec\_t &yerr, size\_t npar, vec\_t &par, mat\_t &covar, double &chi2, param\_t &pa, func\_t &fitfun)  
*Fit the data specified in (xdat,ydat) to the function fitfun with the parameters in par.*
- virtual const char \* `type` ()  
*Return string denoting type ("gsl\_fit").*

## Data Fields

- int `max_iter`  
*(default 500)*
- double `epsabs`  
*(default 1.0e-4)*
- double `epsrel`  
*(default 1.0e-4)*
- bool `test_gradient`  
*If true, test the gradient also (default false).*
- bool `use_scaled`  
*Use the scaled routine if true (default true).*

## Protected Member Functions

- virtual int `print_iter` (int nv, gsl\_vector \*x, gsl\_vector \*dx, int iter, double l\_epsabs, double l\_epsrel)  
*Print the progress in the current iteration.*

## Static Protected Member Functions

- static int [func](#) (const gsl\_vector \*x, void \*pa, gsl\_vector \*f)  
*Evaluate the function.*
- static int [dfunc](#) (const gsl\_vector \*x, void \*pa, gsl\_matrix \*jac)  
*Evaluate the [jacobian](#).*
- static int [fdfunc](#) (const gsl\_vector \*x, void \*pa, gsl\_vector \*f, gsl\_matrix \*jac)  
*Evaluate the function and the [jacobian](#).*

## Data Structures

- struct [func\\_par](#)  
*A structure for passing to the functions [func\(\)](#), [dfunc\(\)](#), and [fdfunc\(\)](#).*

### 3.102.2 Member Function Documentation

**3.102.2.1 virtual int fit (size\_t ndat, vec\_t & xdat, vec\_t & ydat, vec\_t & yerr, size\_t npar, vec\_t & par, mat\_t & covar, double & chi2, param\_t & pa, func\_t & fitfun)** [inline, virtual]

Fit the data specified in (xdat,ydat) to the function fitfun with the parameters in par.

The covariance matrix for the parameters is returned in covar and the value of  $\chi^2$  is returned in chi2.

Reimplemented from [fit\\_base](#).

Definition at line 88 of file gsl\_fit.h.

The documentation for this class was generated from the following file:

- [gsl\\_fit.h](#)

## 3.103 gsl\_fit::func\_par Struct Reference

```
#include <gsl_fit.h>
```

### 3.103.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view, class bool_vec_t = bool *>
struct gsl_fit< param_t, func_t, vec_t, mat_t, bool_vec_t >::func_par
```

A structure for passing to the functions [func\(\)](#), [dfunc\(\)](#), and [fdfunc\(\)](#).

Definition at line 219 of file gsl\_fit.h.

## Data Fields

- func\_t & [f](#)  
*The function object.*
- param\_t \* [vp](#)  
*The user-specified parameter.*
- int [ndat](#)  
*The number.*
- vec\_t \* [xdat](#)  
*The x values.*
- vec\_t \* [ydat](#)  
*The y values.*



- `vec_t * yerr`  
*The y uncertainties.*
- `int npar`  
*The number of parameters.*

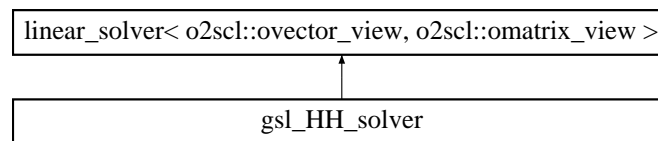
The documentation for this struct was generated from the following file:

- `gsl_fit.h`

## 3.104 gsl\_HH\_solver Class Reference

```
#include <ode_it_solve.h>
```

Inheritance diagram for `gsl_HH_solver`:



### 3.104.1 Detailed Description

GSL Householder solver.

Definition at line 176 of file `ode_it_solve.h`.

#### Public Member Functions

- virtual `int solve` (`size_t n`, `o2scl::omatrix_view &A`, `o2scl::ovector_view &b`, `o2scl::ovector_view &x`)  
*Solve square linear system  $Ax = b$  of size  $n$ .*
- virtual `~gsl_HH_solver` ()

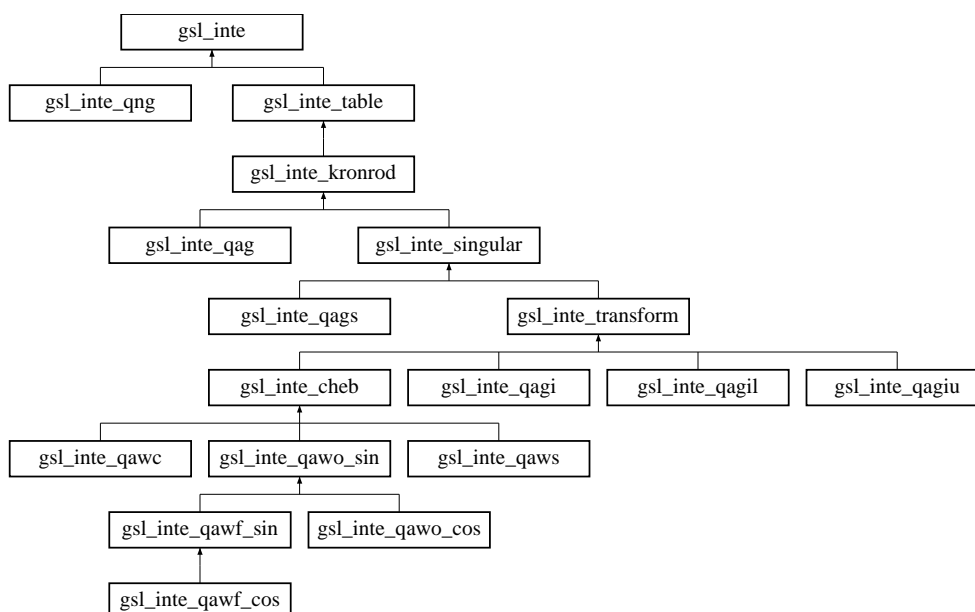
The documentation for this class was generated from the following file:

- `ode_it_solve.h`

## 3.105 gsl\_inte Class Reference

```
#include <gsl_inte.h>
```

Inheritance diagram for `gsl_inte`:



### 3.105.1 Detailed Description

GSL integration base.

This base class does not perform any actual integration.

#### Todo

Move the documentation below to a more sensible place

If the function `integ()` uses `tolx` and `tolf` to attempt to ensure that

$$|\text{result} - I| \leq \text{Max}(\text{tolx}, \text{tolf}|I|)$$

and returns a value `abserr` to attempt to ensure that

$$|\text{result} - I| \leq \text{abserr} \leq \text{Max}(\text{tolx}, \text{tolf}|I|)$$

where  $I$  is the integral to be evaluated. Even when `integ()` returns success, these inequalities will fail for some functions.

Definition at line 54 of file `gsl_inte.h`.

#### Public Member Functions

- [gsl\\_inte\(\)](#)

#### Protected Member Functions

- double [rescale\\_error](#) (double err, const double result\_abs, const double result\_asc)  
*Desc.*

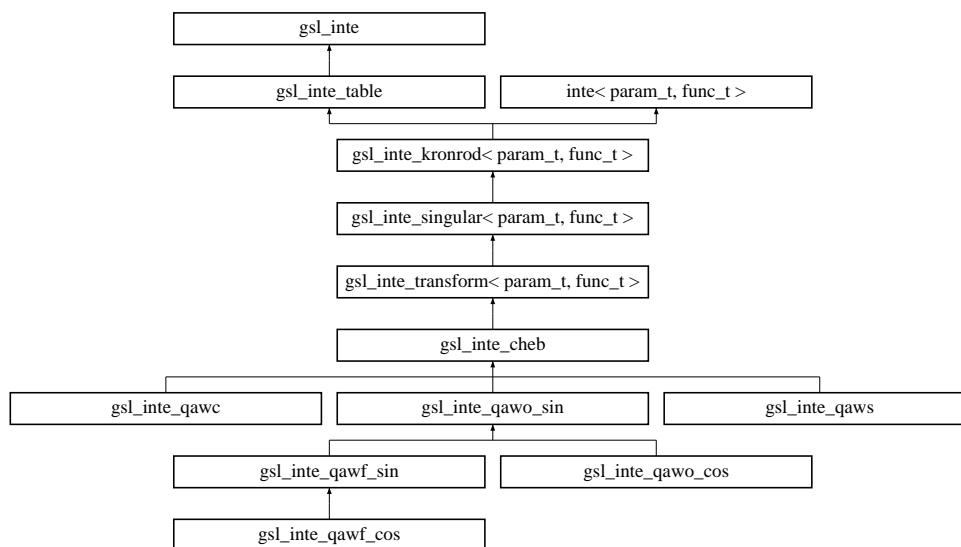
The documentation for this class was generated from the following file:

- `gsl_inte.h`

## 3.106 gsl\_inte\_cheb Class Template Reference

```
#include <gsl_inte_qawc.h>
```

Inheritance diagram for gsl\_inte\_cheb::



### 3.106.1 Detailed Description

**template<class param\_t, class func\_t> class gsl\_inte\_cheb< param\_t, func\_t >**

Integration with Chebyshev (GSL).

Definition at line 35 of file gsl\_inte\_qawc.h.

#### Public Member Functions

- void [compute\\_moments](#) (double cc, double \*moment)  
*Desc.*
- void [gsl\\_integration\\_qcheb](#) (func\_t &f, double a, double b, double \*cheb12, double \*cheb24, param\_t &pa)  
*Desc.*

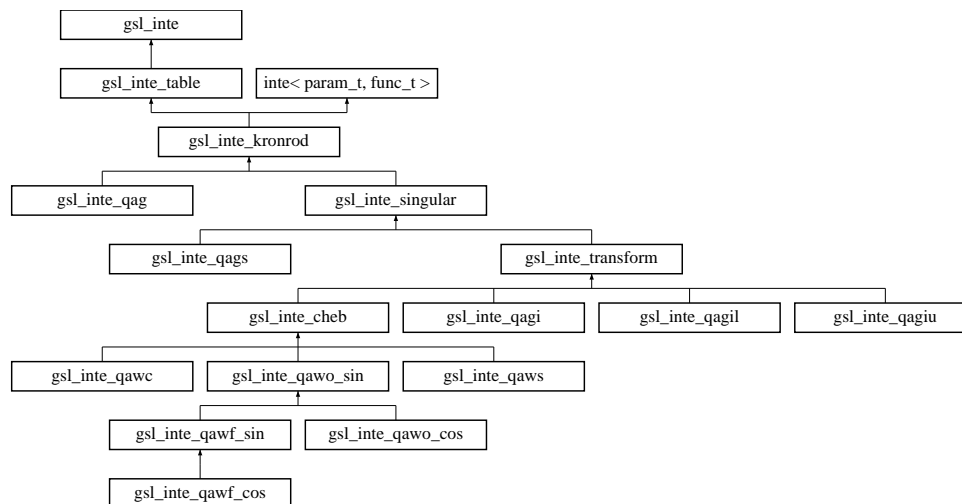
The documentation for this class was generated from the following file:

- gsl\_inte\_qawc.h

## 3.107 gsl\_inte\_kronrod Class Template Reference

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for gsl\_inte\_kronrod::



### 3.107.1 Detailed Description

**template<class param\_t, class func\_t> class gsl\_inte\_kronrod< param\_t, func\_t >**

Desc.

Definition at line 544 of file `gsl_inte_qag_b.h`.

#### Public Member Functions

- virtual void [gsl\\_integration\\_qk\\_o2scl](#) (func\_t &func, const int n, const double xgk[], const double wg[], const double wgk[], double fv1[], double fv2[], double a, double b, double \*result, double \*abserr, double \*resabs, double \*resasc, param\_t &pa)  
*The basic GSL Gauss-Kronrod integration function.*

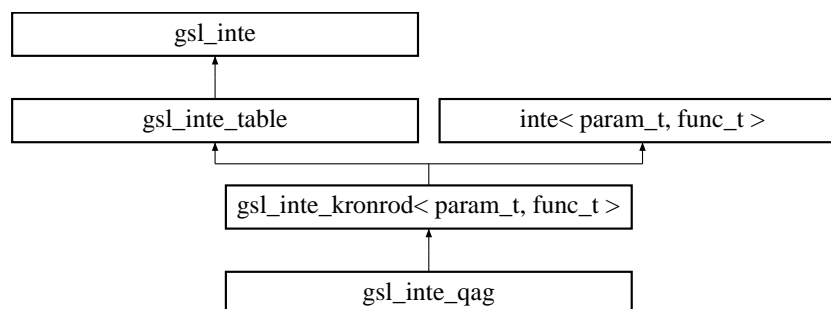
The documentation for this class was generated from the following file:

- `gsl_inte_qag_b.h`

## 3.108 gsl\_inte\_qag Class Template Reference

`#include <gsl_inte_qag.h>`

Inheritance diagram for `gsl_inte_qag::`



### 3.108.1 Detailed Description

**template<class param\_t, class func\_t> class gsl\_inte\_qag< param\_t, func\_t >**

Adaptive integration a function with finite limits of integration (GSL).

#### Todo

Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

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

#### Public Member Functions

- [gsl\\_inte\\_qag](#) ()
- virtual [~gsl\\_inte\\_qag](#) ()
- int [set\\_key](#) (int key)  
*Set the number of integration points.*
- int [get\\_key](#) ()  
*Return the key used (1-6).*
- virtual double [integ](#) (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function func from a to b.*
- virtual int [integ\\_err](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function func from a to b and place the result in res and the error in err.*

#### Protected Member Functions

- int [qag](#) (func\_t &func, const int qn, const double xgk[], const double wg[], const double wgk[], double fv1[], double fv2[], const double a, const double b, const double l\_epsabs, const double l\_epsrel, const size\_t limit, double \*result, double \*abserr, param\_t &pa)  
*Perform an adaptive integration given the coefficients, and returning result.*
- const char \* [type](#) ()  
*Return string denoting type ("gsl\_inte\_qag").*

#### Protected Attributes

- int [lkey](#)  
*Select the number of integration points.*

### 3.108.2 Member Function Documentation

#### 3.108.2.1 int set\_key (int key) [inline]

Set the number of integration points.

The possible values for `key` are:

- 1: `GSL_INTEG_GAUSS15` (default)
  - 2: `GSL_INTEG_GAUSS21`
  - 3: `GSL_INTEG_GAUSS31`
  - 4: `GSL_INTEG_GAUSS41`
  - 5: `GSL_INTEG_GAUSS51`
-

- 6: GSL\_INTEG\_GAUSS61

If an integer other than 1-6 is given, the default (GSL\_INTEG\_GAUSS15) is assumed, and the error handler is called.

Definition at line 73 of file `gsl_inte_qag.h`.

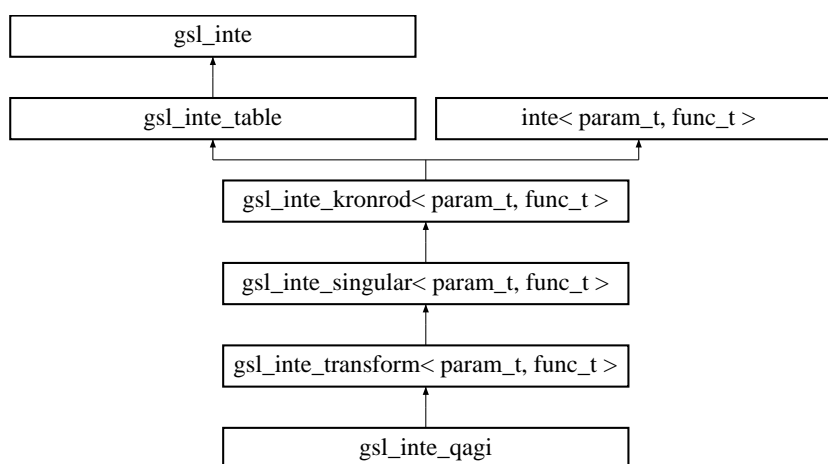
The documentation for this class was generated from the following file:

- `gsl_inte_qag.h`

## 3.109 gsl\_inte\_qagi Class Template Reference

```
#include <gsl_inte_qagi.h>
```

Inheritance diagram for `gsl_inte_qagi`:



### 3.109.1 Detailed Description

**template<class param\_t, class func\_t> class `gsl_inte_qagi`< param\_t, func\_t >**

Integrate a function from  $-\infty$  to  $\infty$  (GSL).

Definition at line 36 of file `gsl_inte_qagi.h`.

#### Public Member Functions

- virtual double `integ` (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function func from  $-\infty$  to  $\infty$ .*
- virtual int `integ_err` (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function func from  $-\infty$  to  $\infty$  giving result res and error err.*

#### Protected Member Functions

- virtual double `transform` (func\_t &func, double t, param\_t &pa)  
*Transformation to  $t \in (0, 1]$ .*

### 3.109.2 Member Function Documentation

#### 3.109.2.1 virtual double integ (func\_t &func, double a, double b, param\_t &pa) [inline, virtual]

Integrate function `func` from  $-\infty$  to  $\infty$ .

The values given in `a` and `b` are ignored

Reimplemented from [inte](#).

Definition at line 46 of file `gsl_inte_qagi.h`.

#### 3.109.2.2 virtual int integ\_err (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2) [inline, virtual]

Integrate function `func` from  $-\infty$  to  $\infty$  giving result `res` and error `err`.

The values `a` and `b` are ignored

Reimplemented from [inte](#).

Definition at line 58 of file `gsl_inte_qagi.h`.

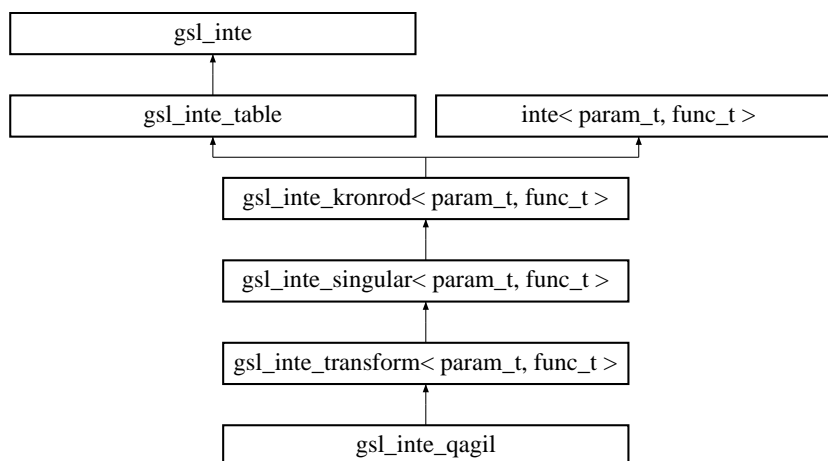
The documentation for this class was generated from the following file:

- `gsl_inte_qagi.h`

## 3.110 gsl\_inte\_qagil Class Template Reference

```
#include <gsl_inte_qagil.h>
```

Inheritance diagram for `gsl_inte_qagil`:



### 3.110.1 Detailed Description

**template<class param\_t, class func\_t> class gsl\_inte\_qagil< param\_t, func\_t >**

Integrate a function from  $-\infty$  to `b` (GSL).

Definition at line 36 of file `gsl_inte_qagil.h`.

## Public Member Functions

- virtual double [integ](#) (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function func from  $-\infty$  to b.*
- virtual int [integ\\_err](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function func from  $-\infty$  to b and place the result in res and the error in err2.*

## Protected Member Functions

- virtual double [transform](#) (func\_t &func, double t, param\_t &pa)  
*Transform to  $t \in (0, 1]$ .*

## Protected Attributes

- double [lb](#)  
*Store the upper limit.*

### 3.110.2 Member Function Documentation

#### 3.110.2.1 virtual double integ (func\_t &func, double a, double b, param\_t &pa) [inline, virtual]

Integrate function func from  $-\infty$  to b.

The value given in a is ignored.

Reimplemented from [inte](#).

Definition at line 55 of file gsl\_inte\_qagil.h.

#### 3.110.2.2 virtual int integ\_err (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2) [inline, virtual]

Integrate function func from  $-\infty$  to b and place the result in res and the error in err2.

The value given in a is ignored.

Reimplemented from [inte](#).

Definition at line 68 of file gsl\_inte\_qagil.h.

The documentation for this class was generated from the following file:

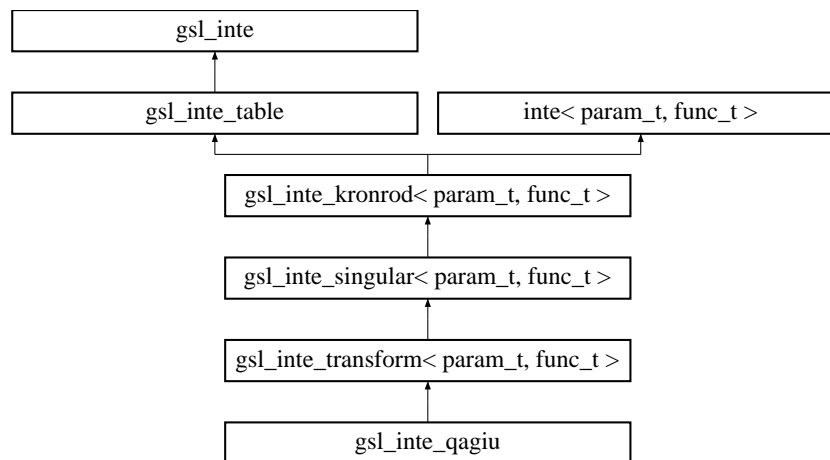
- [gsl\\_inte\\_qagil.h](#)

## 3.111 gsl\_inte\_qagiu Class Template Reference

```
#include <gsl_inte_qagiu.h>
```

Inheritance diagram for gsl\_inte\_qagiu::





### 3.111.1 Detailed Description

**template<class param\_t, class func\_t> class gsl\_inte\_qagiu< param\_t, func\_t >**

Integrate a function from  $a$  to  $\infty$  (GSL).

#### Todo

I had to add extra code to check for non-finite values for some integrations. This should be checked.

The extra line was of the form:

```
if (!finite(areal)) areal=0.0;
```

Definition at line 44 of file `gsl_inte_qagiu.h`.

#### Public Member Functions

- virtual double [integ](#) (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function func from a to  $\infty$ .*
- virtual int [integ\\_err](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function func from a to  $\infty$  giving result res and error err.*

#### Protected Member Functions

- virtual double [transform](#) (func\_t &func, double t, param\_t &pa)  
*Transform to  $t \in (0, 1]$ .*

#### Protected Attributes

- double [la](#)  
*Store the lower limit.*

### 3.111.2 Member Function Documentation

#### 3.111.2.1 virtual double integ (func\_t &func, double a, double b, param\_t &pa) [inline, virtual]

Integrate function `func` from `a` to  $\infty$ .

The value `b` is ignored.

Reimplemented from [inte](#).

Definition at line 64 of file `gsl_inte_qagiu.h`.

#### 3.111.2.2 virtual int integ\_err (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2) [inline, virtual]

Integrate function `func` from `a` to  $\infty$  giving result `res` and error `err`.

The value `b` is ignored.

Reimplemented from [inte](#).

Definition at line 77 of file `gsl_inte_qagiu.h`.

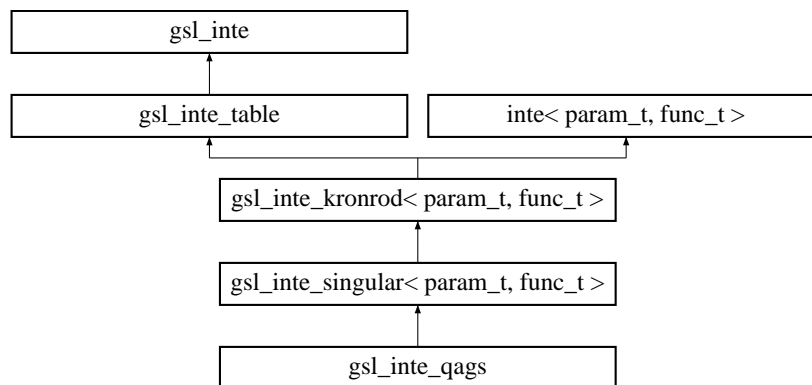
The documentation for this class was generated from the following file:

- `gsl_inte_qagiu.h`

## 3.112 gsl\_inte\_qags Class Template Reference

```
#include <gsl_inte_qags.h>
```

Inheritance diagram for `gsl_inte_qags`:



### 3.112.1 Detailed Description

```
template<class param_t, class func_t> class gsl_inte_qags< param_t, func_t >
```

Integrate a function with a singularity (GSL).

Definition at line 35 of file `gsl_inte_qags.h`.

#### Public Member Functions

- virtual double [integ](#) (func\_t &func, double a, double b, param\_t &pa)

*Integrate function `func` from `a` to `b`.*

- virtual int [integ\\_err](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.*

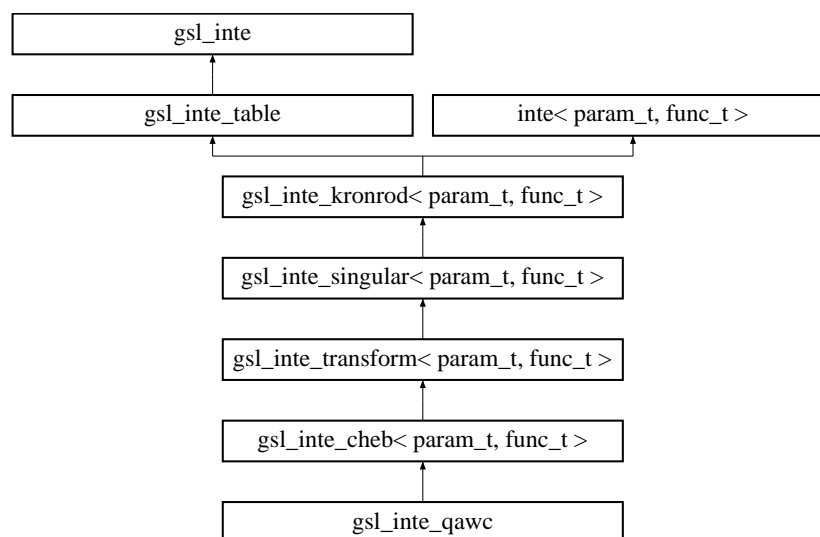
The documentation for this class was generated from the following file:

- gsl\_inte\_qags.h

### 3.113 gsl\_inte\_qawc Class Template Reference

```
#include <gsl_inte_qawc.h>
```

Inheritance diagram for `gsl_inte_qawc`:



#### 3.113.1 Detailed Description

**template<class param\_t, class func\_t> class `gsl_inte_qawc`< param\_t, func\_t >**

Adaptive Cauchy principal value integration (GSL).

Definition at line 287 of file `gsl_inte_qawc.h`.

#### Public Member Functions

- [gsl\\_inte\\_qawc](#) ()
- virtual [~gsl\\_inte\\_qawc](#) ()
- virtual double [integ](#) (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function `func` from `a` to `b`.*
- virtual int [integ\\_err](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.*

#### Data Fields

- double [s](#)  
*The singularity.*

## Protected Member Functions

- int [qawc](#) (func\_t &func, const double a, const double b, const double c, const double epsabs, const double epsrel, const size\_t limit, double \*result, double \*abserr, param\_t &pa)  
*Desc.*
- void [qc25c](#) (func\_t &func, double a, double b, double c, double \*result, double \*abserr, int \*err\_reliable, param\_t &pa)  
*Desc.*
- virtual double [transform](#) (func\_t &func, double x, param\_t &pa)  
*Desc.*
- const char \* [type](#) ()  
*Return string denoting type ("gsl\_inte\_qawc").*

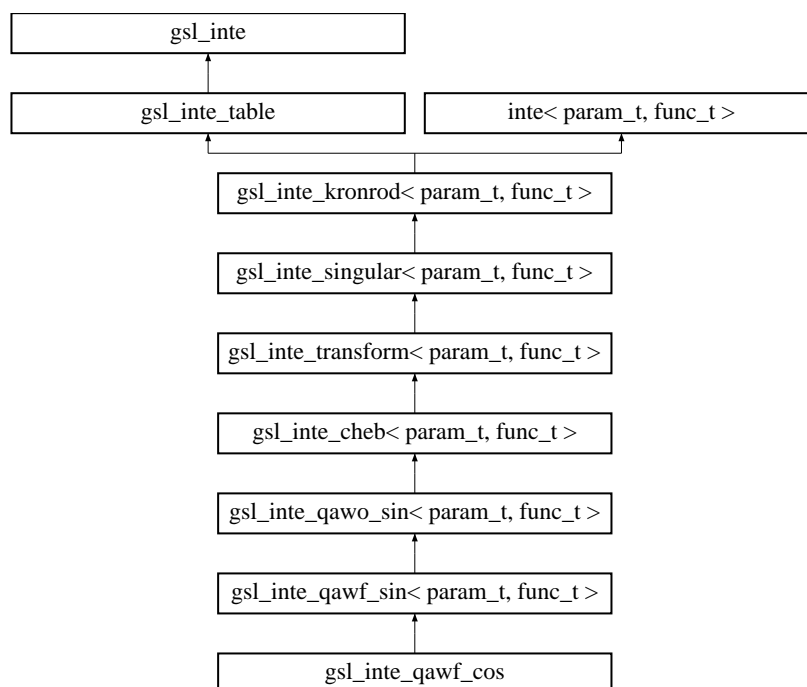
The documentation for this class was generated from the following file:

- gsl\_inte\_qawc.h

## 3.114 gsl\_inte\_qawf\_cos Class Template Reference

```
#include <gsl_inte_qawf.h>
```

Inheritance diagram for gsl\_inte\_qawf\_cos::



### 3.114.1 Detailed Description

```
template<class param_t, class func_t> class gsl_inte_qawf_cos< param_t, func_t >
```

Adaptive integration a function with finite limits of integration (GSL).

#### Todo

Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Definition at line 353 of file gsl\_inte\_qawf.h.

## Public Member Functions

- [gsl\\_inte\\_qawf\\_cos](#) ()
- virtual [~gsl\\_inte\\_qawf\\_cos](#) ()
- virtual int [integ\\_err](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function func from a to b and place the result in res and the error in err.*

## Protected Member Functions

- virtual double [transform](#) (func\_t &func, double x, param\_t &pa)  
*Desc.*
- const char \* [type](#) ()  
*Return string denoting type ("gsl\_inte\_qawf\_cos").*

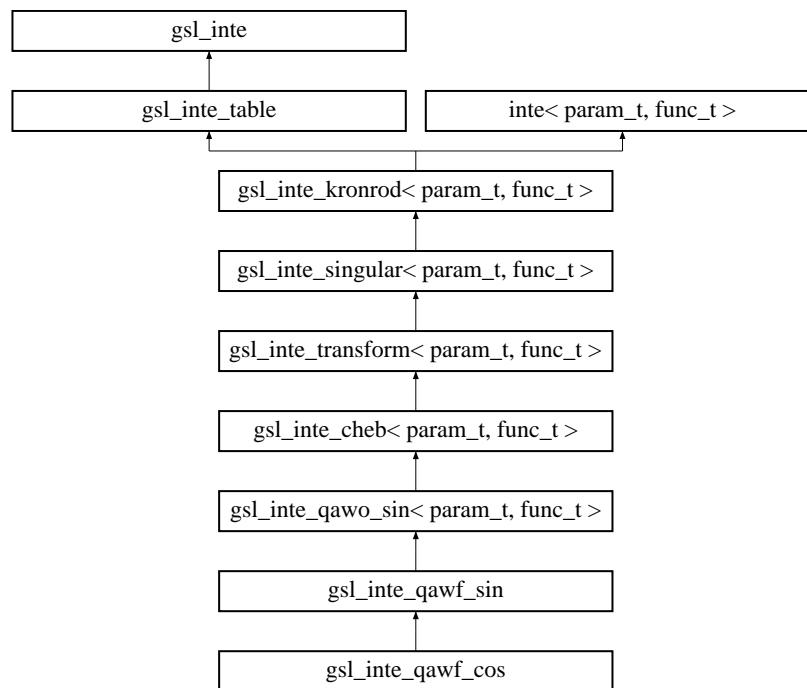
The documentation for this class was generated from the following file:

- [gsl\\_inte\\_qawf.h](#)

## 3.115 gsl\_inte\_qawf\_sin Class Template Reference

```
#include <gsl_inte_qawf.h>
```

Inheritance diagram for `gsl_inte_qawf_sin`:



### 3.115.1 Detailed Description

```
template<class param_t, class func_t> class gsl_inte_qawf_sin< param_t, func_t >
```

Adaptive integration for oscillatory integrals (GSL).

Definition at line 36 of file `gsl_inte_qawf.h`.

## Public Member Functions

- [gsl\\_inte\\_qawf\\_sin](#) ()
- virtual [~gsl\\_inte\\_qawf\\_sin](#) ()
- virtual double [integ](#) (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function func from a to b.*
- virtual int [integ\\_err](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function func from a to b and place the result in res and the error in err.*

## Protected Member Functions

- int [qawf](#) (func\_t &func, const double a, const double epsabs, const size\_t limit, double \*result, double \*abserr, param\_t &pa)  
*Desc.*
- virtual double [transform](#) (func\_t &func, double x, param\_t &pa)  
*Desc.*
- const char \* [type](#) ()  
*Return string denoting type ("gsl\_inte\_qawf\_sin").*

## Protected Attributes

- gsl\_integration\_workspace \* [cyclew](#)  
*Desc.*

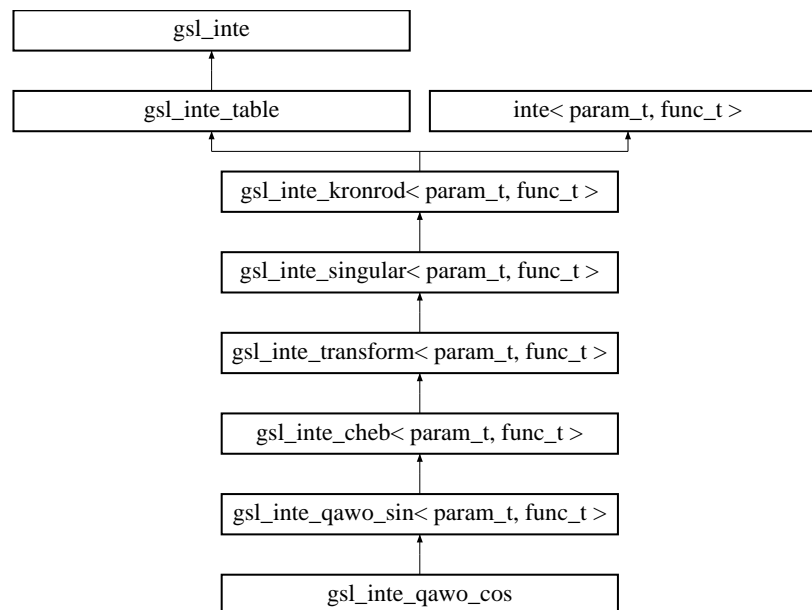
The documentation for this class was generated from the following file:

- [gsl\\_inte\\_qawf.h](#)

## 3.116 gsl\_inte\_qawo\_cos Class Template Reference

```
#include <gsl_inte_qawo.h>
```

Inheritance diagram for `gsl_inte_qawo_cos`:



### 3.116.1 Detailed Description

```
template<class param_t, class func_t> class gsl_inte_qawo_cos< param_t, func_t >
```

Adaptive integration a function with finite limits of integration (GSL).

#### Todo

Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Definition at line 637 of file `gsl_inte_qawo.h`.

#### Public Member Functions

- `gsl_inte_qawo_cos()`
- virtual `~gsl_inte_qawo_cos()`
- virtual int `integ_err` (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function func from a to b and place the result in res and the error in err.*

#### Protected Member Functions

- virtual double `transform` (func\_t &func, double x, param\_t &pa)  
*Desc.*
- const char \* `type` ()  
*Return string denoting type ("gsl\_inte\_qawo\_cos").*

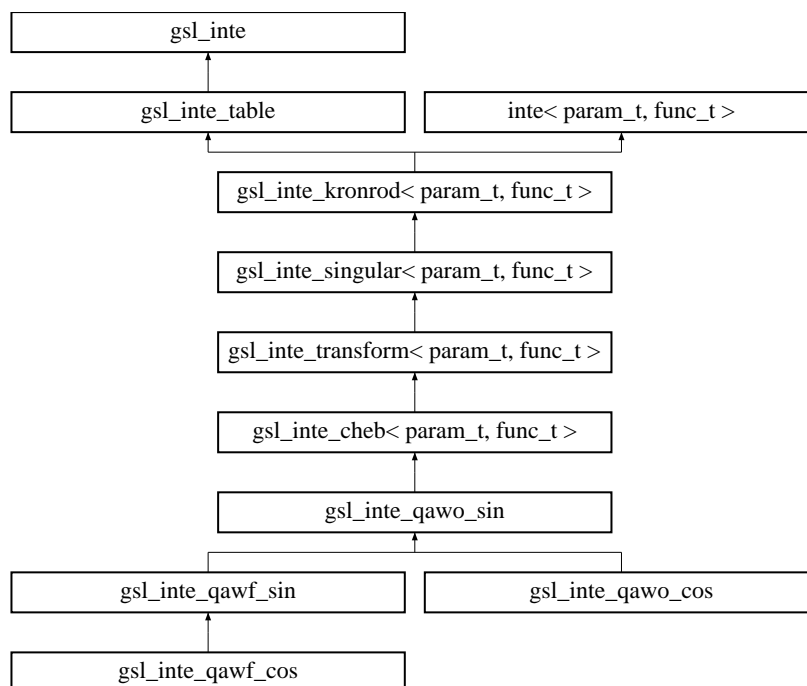
The documentation for this class was generated from the following file:

- `gsl_inte_qawo.h`

## 3.117 `gsl_inte_qawo_sin` Class Template Reference

```
#include <gsl_inte_qawo.h>
```

Inheritance diagram for `gsl_inte_qawo_sin`:



### 3.117.1 Detailed Description

**template<class param\_t, class func\_t> class `gsl_inte_qawo_sin`< param\_t, func\_t >**

Adaptive integration for oscillatory integrals (GSL).

Definition at line 35 of file `gsl_inte_qawo.h`.

#### Public Member Functions

- [`gsl\_inte\_qawo\_sin`](#) ()
- virtual [`~gsl\_inte\_qawo\_sin`](#) ()
- virtual double [`integ`](#) (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function `func` from `a` to `b`.*
- virtual int [`integ\_err`](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.*

#### Data Fields

- double [`omega`](#)  
*Desc.*
- size\_t [`tab\_size`](#)  
*Desc.*

#### Protected Member Functions

- int [`qawo`](#) (func\_t &func, const double a, const double epsabs, const double epsrel, const size\_t limit, gsl\_integration\_workspace \*loc\_w, gsl\_integration\_qawo\_table \*wf, double \*result, double \*abserr, param\_t &pa)  
*Desc.*
- void [`qc25f`](#) (func\_t &func, double a, double b, gsl\_integration\_qawo\_table \*wf, size\_t level, double \*result, double \*abserr, double \*resabs, double \*resasc, param\_t &pa)



*Desc.*

- virtual double [transform](#) (func\_t &func, double x, param\_t &pa)

*Desc.*

- const char \* [type](#) ()

*Return string denoting type ("gsl\_inte\_qawo\_sin").*

### Protected Attributes

- gsl\_integration\_qawo\_table \* [otable](#)

*Desc.*

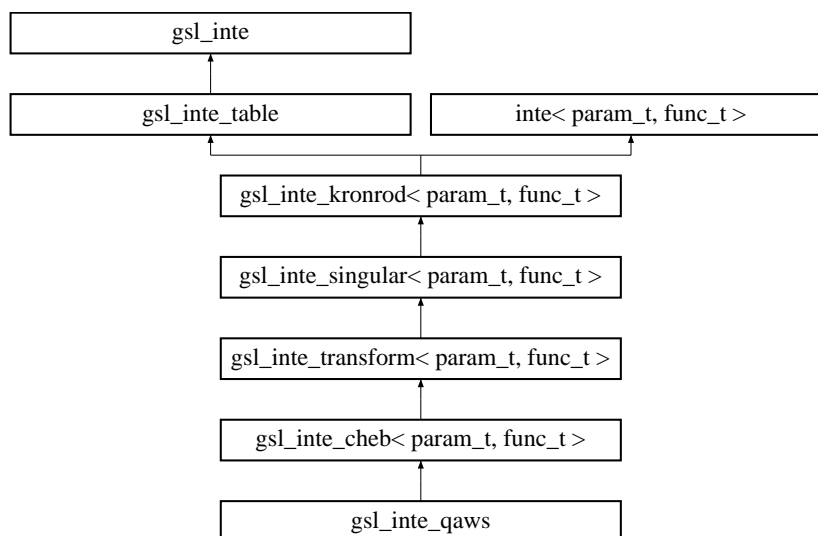
The documentation for this class was generated from the following file:

- gsl\_inte\_qawo.h

## 3.118 gsl\_inte\_qaws Class Template Reference

```
#include <gsl_inte_qaws.h>
```

Inheritance diagram for gsl\_inte\_qaws::



### 3.118.1 Detailed Description

```
template<class param_t, class func_t> class gsl_inte_qaws< param_t, func_t >
```

Adaptive Cauchy principal value integration (GSL).

Definition at line 36 of file gsl\_inte\_qaws.h.

### Public Member Functions

- [gsl\\_inte\\_qaws](#) ()
- virtual [~gsl\\_inte\\_qaws](#) ()
- virtual double [integ](#) (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function func from a to b.*
- virtual int [integ\\_err](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err2)  
*Integrate function func from a to b and place the result in res and the error in err.*

## Data Fields

- double [s](#)  
*The singularity.*

## Protected Member Functions

- int [qaws](#) (func\_t &func, const double a, const double b, const double c, const double epsabs, const double epsrel, const size\_t limit, double \*result, double \*abserr, param\_t &pa)  
*Desc.*
- virtual double [transform](#) (func\_t &func, double x, param\_t &pa)  
*Desc.*
- const char \* [type](#) ()  
*Return string denoting type ("gsl\_inte\_qaws").*

## Static Protected Member Functions

- static double [fn\\_qaws](#) (double t, void \*params)
- static double [fn\\_qaws\\_L](#) (double x, void \*params)
- static double [fn\\_qaws\\_R](#) (double x, void \*params)
- static void [compute\\_result](#) (const double \*r, const double \*cheb12, const double \*cheb24, double \*result12, double \*result24)
- static void [qc25s](#) (gsl\_function \*f, double a, double b, double a1, double b1, gsl\_integration\_qaws\_table \*t, double \*result, double \*abserr, int \*err\_reliable)
- static void [qc25s](#) (gsl\_function \*f, double a, double b, double a1, double b1, gsl\_integration\_qaws\_table \*t, double \*result, double \*abserr, int \*err\_reliable)
- static double [fn\\_qaws](#) (double x, void \*params)
- static double [fn\\_qaws\\_L](#) (double x, void \*params)
- static double [fn\\_qaws\\_R](#) (double x, void \*params)
- static void [compute\\_result](#) (const double \*r, const double \*cheb12, const double \*cheb24, double \*result12, double \*result24)

## Data Structures

- struct [fn\\_qaws\\_params](#)

The documentation for this class was generated from the following file:

- [gsl\\_inte\\_qaws.h](#)

## 3.119 gsl\_inte\_qaws::fn\_qaws\_params Struct Reference

### 3.119.1 Detailed Description

```
template<class param_t, class func_t> struct gsl_inte_qaws< param_t, func_t >::fn_qaws_params
```

Definition at line 269 of file [gsl\\_inte\\_qaws.h](#).

## Data Fields

- gsl\_function \* [function](#)
- double [a](#)
- double [b](#)

- `gsl_integration_qaws_table * table`

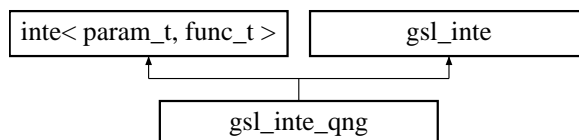
The documentation for this struct was generated from the following file:

- `gsl_inte_qaws.h`

## 3.120 gsl\_inte\_qng Class Template Reference

```
#include <gsl_inte_qng.h>
```

Inheritance diagram for `gsl_inte_qng`:



### 3.120.1 Detailed Description

`template<class param_t, class func_t> class gsl_inte_qng< param_t, func_t >`

Non-adaptive integration from a to b (GSL).

`integ()` uses 10-point, 21-point, 43-point, and 87-point Gauss-Kronrod integration successively until the integral is returned within the accuracy specified by `tolx` and `tolf`.

Definition at line 224 of file `gsl_inte_qng.h`.

#### Public Member Functions

- virtual double `integ` (`func_t &func`, double `a`, double `b`, `param_t &pa`)  
*Integrate function `func` from a to b.*
- virtual int `integ_err` (`func_t &func`, double `a`, double `b`, `param_t &pa`, double `&res`, double `&err2`)  
*Integrate function `func` from a to b giving result `res` and error `err`.*
- const char \* `type` ()  
*Return string denoting type ("gsl\_inte\_qng").*

#### Data Fields

- size\_t `feval`  
*The number of function evaluations for the last integration.*

### 3.120.2 Field Documentation

#### 3.120.2.1 size\_t feval

The number of function evaluations for the last integration.

Set to either 0, 21, 43, or 87, depending on the number of function evaluations that were used. This variable is zero if an error occurs before any function evaluations were performed and is never equal 10, since in the 10-point method, the 21-point result is used to estimate the error. If the function fails to achieve the desired precision, `feval` is set to 88.

Definition at line 239 of file `gsl_inte_qng.h`.

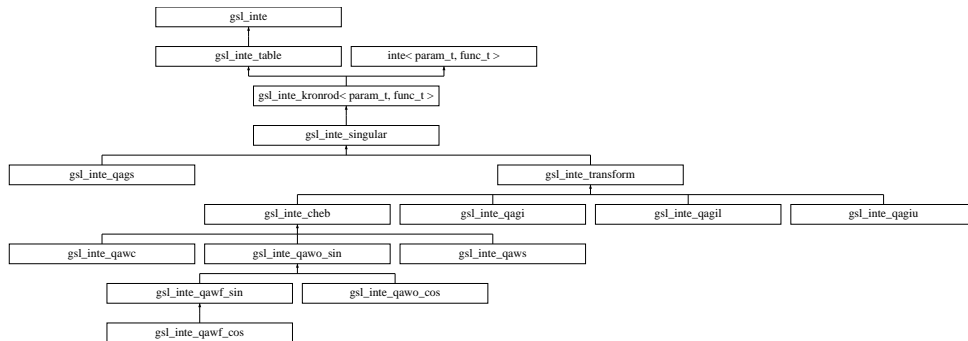
The documentation for this class was generated from the following file:

- `gsl_inte_qng.h`

## 3.121 gsl\_inte\_singular Class Template Reference

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for `gsl_inte_singular`:



### 3.121.1 Detailed Description

**template<class param\_t, class func\_t> class `gsl_inte_singular`< param\_t, func\_t >**

Integrate a function with a singularity (GSL).

Definition at line 638 of file `gsl_inte_qag_b.h`.

#### Protected Member Functions

- void `initialise_table` (struct `extrapolation_table` \*table)  
*Desc.*
- void `append_table` (struct `extrapolation_table` \*table, double y)  
*Desc.*
- int `test_positivity` (double result, double resabs)  
*Desc.*
- void `qelg` (struct `extrapolation_table` \*table, double \*result, double \*abserr)  
*Desc.*
- int `large_interval` (gsl\_integration\_workspace \*workspace)  
*Desc.*
- void `reset_nrmx` (gsl\_integration\_workspace \*workspace)  
*Desc.*
- int `increase_nrmx` (gsl\_integration\_workspace \*workspace)  
*Desc.*
- int `qags` (func\_t &func, const int qn, const double xgk[ ], const double wg[ ], const double wgl[ ], double fv1[ ], double fv2[ ], const double a, const double b, const double l\_epsabs, const double l\_epsrel, const size\_t limit, double \*result, double \*abserr, param\_t &pa)  
*Desc.*

#### Data Structures

- struct `extrapolation_table`  
*A structure for extrapolation for `gsl_inte_qags`.*

The documentation for this class was generated from the following file:

- `gsl_inte_qag_b.h`

## 3.122 `gsl_inte_singular::extrapolation_table` Struct Reference

```
#include <gsl_inte_qag_b.h>
```

### 3.122.1 Detailed Description

**template<class param\_t, class func\_t> struct `gsl_inte_singular< param_t, func_t >::extrapolation_table`**

A structure for extrapolation for [gsl\\_inte\\_qags](#).

#### Todo

Move this to a new class, with [qe1g\(\)](#) as a method

Definition at line 648 of file `gsl_inte_qag_b.h`.

#### Data Fields

- `size_t` [n](#)
- `double` [rlist2](#) [52]
- `size_t` [nres](#)
- `double` [res3la](#) [3]

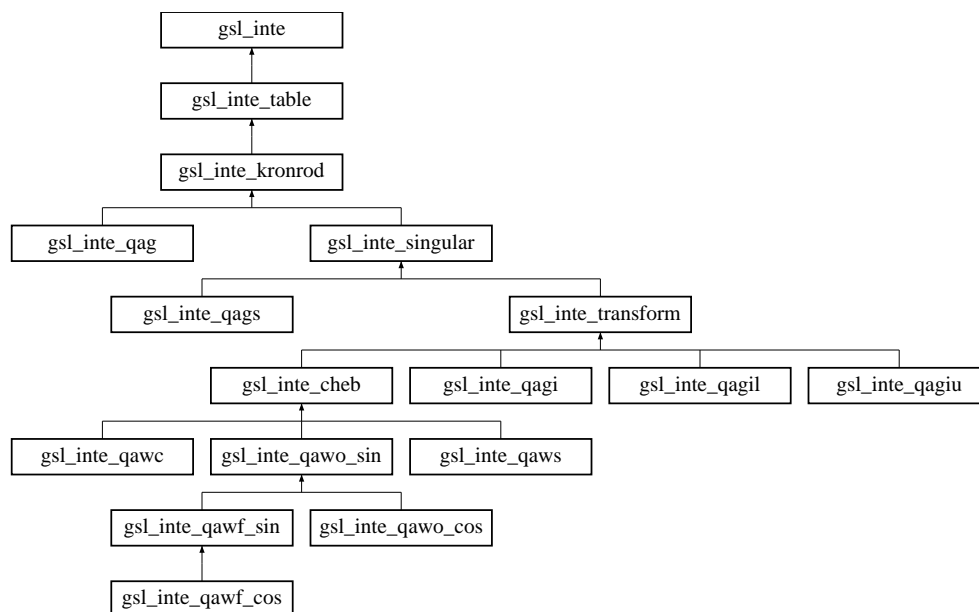
The documentation for this struct was generated from the following file:

- `gsl_inte_qag_b.h`

## 3.123 `gsl_inte_table` Class Reference

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for `gsl_inte_table`:



### 3.123.1 Detailed Description

Base routines for the GSL adaptive integration routines.

#### Todo

Move `gsl_integration_workspace` to a separate class and remove this class, making all children direct descendants of `gsl_inte` instead. We'll have to figure out what to do with the data member `workspace` though.

Definition at line 485 of file `gsl_inte_qag.b.h`.

#### Public Member Functions

- `int set_workspace (size_t size)`  
*Set the integration workspace size.*
- `void initialise (gsl_integration_workspace *workspace, double a, double b)`  
*Initialize the workspace for an integration with limits `a` and `b`.*
- `void set_initial_result (gsl_integration_workspace *workspace, double result, double error)`  
*Set the result at position zero.*
- `void retrieve (const gsl_integration_workspace *workspace, double *a, double *b, double *r, double *e)`  
*Retrieve the `i`th result from the workspace.*
- `void qpsrt (gsl_integration_workspace *workspace)`  
*Sort the workspace.*
- `void update (gsl_integration_workspace *workspace, double a1, double b1, double area1, double error1, double a2, double b2, double area2, double error2)`  
*Update workspace with new results and resort.*
- `double sum_results (const gsl_integration_workspace *workspace)`  
*Add up all of the contributions to construct the final result.*
- `int subinterval_too_small (double a1, double a2, double b2)`  
*Find out if the present subinterval is too small.*
- `void append_interval (gsl_integration_workspace *workspace, double a1, double b1, double area1, double error1)`  
*Append new results to workspace.*

## Data Fields

- `gsl_integration_workspace * w`  
*The integration workspace.*
- `int wkspc`  
*The size of the integration workspace.*

### 3.123.2 Member Function Documentation

#### 3.123.2.1 void retrieve (const gsl\_integration\_workspace \* workspace, double \* a, double \* b, double \* r, double \* e)

Retrieve the *i*th result from the workspace.

The workspace variable *i* is used to specify which interval is requested.

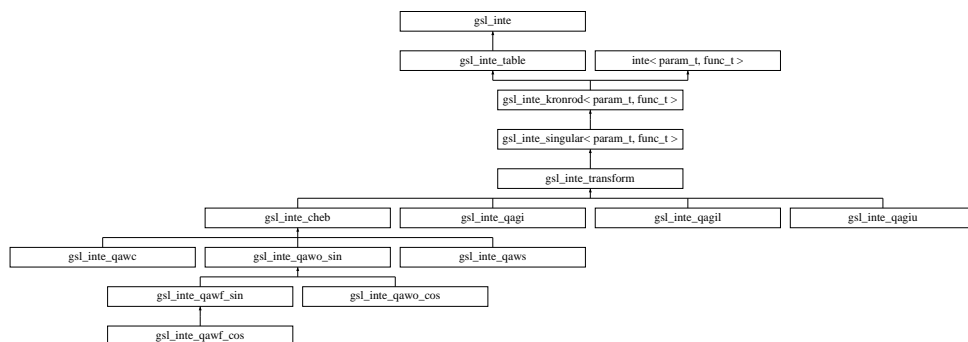
The documentation for this class was generated from the following file:

- `gsl_inte_qag_b.h`

## 3.124 gsl\_inte\_transform Class Template Reference

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for `gsl_inte_transform`:



### 3.124.1 Detailed Description

`template<class param_t, class func_t> class gsl_inte_transform< param_t, func_t >`

Integrate a function with a singularity (GSL).

Definition at line 1274 of file `gsl_inte_qag_b.h`.

## Public Member Functions

- virtual double `transform` (func\_t &func, double t, param\_t &pa)
- virtual void `gsl_integration_qk_o2scl` (func\_t &func, const int n, const double xgk[], const double wg[], const double wgk[], double fv1[], double fv2[], double a, double b, double \*result, double \*abserr, double \*resabs, double \*resasc, param\_t &pa)  
*The basic Gauss-Kronrod integration function.*

### 3.124.2 Member Function Documentation

**3.124.2.1** `virtual void gsl_integration_qk_o2scl (func_t & func, const int n, const double xgk[], const double wg[], const double wgk[], double fv1[], double fv2[], double a, double b, double * result, double * abserr, double * resabs, double * resasc, param_t & pa)` [inline, virtual]

The basic Gauss-Kronrod integration function.

This is basically just a copy of `gsl_inte_qag::gsl_integration_qk_o2scl()` which is rewritten to call the internal transformed function rather than directly calling the user-specified function.

Reimplemented from `gsl_inte_kronrod`.

Definition at line 1292 of file `gsl_inte_qag_b.h`.

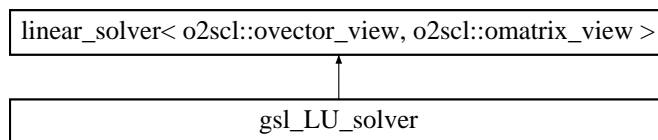
The documentation for this class was generated from the following file:

- `gsl_inte_qag_b.h`

## 3.125 gsl\_LU\_solver Class Reference

```
#include <ode_it_solve.h>
```

Inheritance diagram for `gsl_LU_solver`:



### 3.125.1 Detailed Description

GSL solver by LU decomposition.

Definition at line 133 of file `ode_it_solve.h`.

#### Public Member Functions

- `virtual int solve (size_t n, o2scl::omatrix_view &A, o2scl::ovector_view &b, o2scl::ovector_view &x)`  
Solve square linear system  $Ax = b$  of size  $n$ .
- `virtual ~gsl_LU_solver ()`

The documentation for this class was generated from the following file:

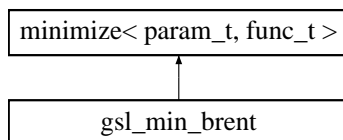
- `ode_it_solve.h`

## 3.126 gsl\_min\_brent Class Template Reference

```
#include <gsl_min_brent.h>
```

Inheritance diagram for `gsl_min_brent`:





### 3.126.1 Detailed Description

**template<class param\_t, class func\_t = funct<param\_t>> class gsl\_min\_brent< param\_t, func\_t >**

One-dimensional minimization (GSL).

#### Todo

Simplify temporary storage and document stopping conditions.

Definition at line 40 of file `gsl_min_brent.h`.

#### Public Member Functions

- int [set](#) (func\_t &func, double xmin, double lower, double upper, param\_t &pa)  
*Set the function to be minimized and the initial bracketing interval.*
- int [set\\_with\\_values](#) (func\_t &func, double xmin, double fmin, double lower, double fl, double upper, double fu, param\_t &pa)  
*Set the function to be minimized and the initial bracketing interval.*
- int [iterate](#) ()  
*Perform an iteration.*
- virtual [~gsl\\_min\\_brent](#) ()
- virtual int [min\\_bkt](#) (double &x2, double x1, double x3, double &fmin, param\_t &pa, func\_t &func)  
*Calculate the minimum min of func with x2 bracketed between x1 and x3.*
- virtual const char \* [type](#) ()  
*Return string denoting type ("gsl\_min\_brent").*

#### Data Fields

- double [x\\_minimum](#)  
*Location of minimum.*
- double [x\\_lower](#)  
*Lower bound.*
- double [x\\_upper](#)  
*Upper mound.*
- double [f\\_minimum](#)  
*Minimum value.*
- double [f\\_lower](#)  
*Value at lower bound.*
- double [f\\_upper](#)  
*Value at upper bound.*

#### Protected Member Functions

- int [compute\\_f\\_values](#) (func\_t &func, double xminimum, double \*fminimum, double xlower, double \*flower, double xupper, double \*fupper, param\_t &pa)  
*Compute the function values at the various points.*

## Protected Attributes

- `func_t * uf`  
*The function.*
- `param_t * up`  
*The parameters.*

## Temporary storage

- `double d`
- `double e`
- `double v`
- `double w`
- `double f_v`
- `double f_w`

### 3.126.2 Member Function Documentation

**3.126.2.1** `virtual int min_bkt(double &x2, double x1, double x3, double &fmin, param_t &pa, func_t &func)` [`inline`, `virtual`]

Calculate the minimum `min` of `func` with `x2` bracketed between `x1` and `x3`.

Note that this algorithm requires that the initial guess already brackets the minimum, i.e.  $x_1 < x_2 < x_3$ ,  $f(x_1) > f(x_2)$  and  $f(x_3) > f(x_2)$ . This is different from `cern_minimize`, where the initial value of the first parameter to `cern_minimize::min_bkt()` is ignored.

Reimplemented from `minimize< param_t, func_t >`.

Definition at line 276 of file `gsl_min_brent.h`.

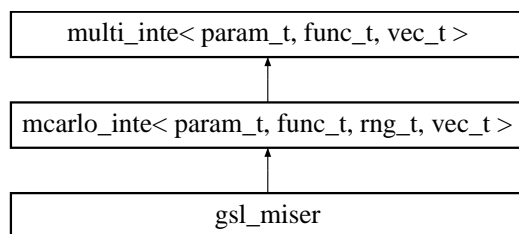
The documentation for this class was generated from the following file:

- `gsl_min_brent.h`

## 3.127 gsl\_miser Class Template Reference

```
#include <gsl_miser.h>
```

Inheritance diagram for `gsl_miser`:



### 3.127.1 Detailed Description

```
template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_view> class gsl_miser< param_t, func_t, rng_t, vec_t >
```

Multidimensional integration using plain Miser Carlo (GSL).

Definition at line 42 of file `gsl_miser.h`.

## Public Member Functions

- virtual [~gsl\\_miser](#) ()
- virtual int [minteg\\_err](#) (func\_t &func, size\_t ndim, const vec\_t &a, const vec\_t &b, param\_t &pa, double &res, double &err)  
*Integrate function func from  $x=a$  to  $x=b$ .*
- virtual const char \* [type](#) ()  
*Return string denoting type ("gsl\_miser").*

## Static Protected Member Functions

- static double [gsl\\_func](#) (double \*x, size\_t dim, void \*pa)  
*The GSL function pointer.*

The documentation for this class was generated from the following file:

- [gsl\\_miser.h](#)

## 3.128 gsl\_mmin\_base Class Template Reference

```
#include <gsl_mmin_conf.h>
```

Inheritance diagram for `gsl_mmin_base`:



### 3.128.1 Detailed Description

```
template<class param_t, class func_t = multi_func<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc, class dfunc_t = grad_func<param_t, ovector_view>, class auto_grad_t = gradient<param_t, func_t,
ovector_view>, class def_auto_grad_t = simple_grad<param_t, func_t, ovector_view>> class gsl_mmin_base< param_t,
func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >
```

Base minimization routines for [gsl\\_mmin\\_conf](#) and [gsl\\_mmin\\_conp](#).

Definition at line 42 of file `gsl_mmin_conf.h`.

## Public Member Functions

- [gsl\\_mmin\\_base](#) ()
- int [base\\_set](#) (func\_t &ufunc, param\_t &pa)  
*Set the function.*
- int [base\\_set\\_de](#) (func\_t &ufunc, dfunc\_t &udfunc, param\_t &pa)  
*Set the function and the [gradient](#).*
- int [base\\_allocate](#) (size\_t nn)  
*Allocate memory.*
- int [base\\_free](#) ()  
*Clear allocated memory.*

## Data Fields

- double [deriv\\_h](#)  
*Stepsize for finite-differencing ( default  $10^{-4}$  ).*

- int `nmaxiter`  
*Maximum iterations for line minimization (default 10).*
- `def_auto_grad_t` `def_grad`  
*Default automatic Gradient object.*

### Protected Member Functions

- void `take_step` (const `gsl_vector` \*`x`, const `gsl_vector` \*`px`, double `stepx`, double `lambda`, `gsl_vector` \*`x1x`, `gsl_vector` \*`dx`)  
*Take a step.*
- void `intermediate_point` (const `gsl_vector` \*`x`, const `gsl_vector` \*`px`, double `lambda`, double `pg`, double `stepa`, double `stepc`, double `fa`, double `fc`, `gsl_vector` \*`x1x`, `gsl_vector` \*`dx`, `gsl_vector` \*`gradient`, double \*`stepx`, double \*`f`)  
*Line minimization.*
- void `minimize` (const `gsl_vector` \*`x`, const `gsl_vector` \*`xp`, double `lambda`, double `stepa`, double `stepb`, double `stepc`, double `fa`, double `fb`, double `fc`, double `xtol`, `gsl_vector` \*`x1x`, `gsl_vector` \*`dx1x`, `gsl_vector` \*`x2x`, `gsl_vector` \*`dx2x`, `gsl_vector` \*`gradient`, double \*`xstep`, double \*`f`, double \*`gnorm_u`)  
*Perform the minimization.*

### Protected Attributes

- `func_t` \* `func`  
*User-specified function.*
- `dfunc_t` \* `grad`  
*User-specified gradient.*
- `auto_grad_t` \* `agrad`  
*Automatic gradient object.*
- bool `grad_given`  
*If true, a gradient has been specified.*
- `param_t` \* `params`  
*User-specified parameter.*
- `size_t` `dim`  
*Memory size.*
- `alloc_t` `ao`  
*Memory allocation.*
- `alloc_vec_t` `avt`  
*Temporary vector.*
- `alloc_vec_t` `avt2`

## 3.128.2 Member Function Documentation

**3.128.2.1** void `intermediate_point` (const `gsl_vector` \*`x`, const `gsl_vector` \*`px`, double `lambda`, double `pg`, double `stepa`, double `stepc`, double `fa`, double `fc`, `gsl_vector` \*`x1x`, `gsl_vector` \*`dx`, `gsl_vector` \*`gradient`, double \*`stepx`, double \*`f`) [`inline`, `protected`]

Line minimization.

Do a line minimisation in the region (xa,fa) (xc,fc) to find an intermediate (xb,fb) satisfying  $fa > fb < fc$ . Choose an initial xb based on parabolic interpolation

Definition at line 89 of file `gsl_mmin_conf.h`.

**3.128.2.2** void `minimize` (const `gsl_vector` \*`x`, const `gsl_vector` \*`xp`, double `lambda`, double `stepa`, double `stepb`, double `stepc`, double `fa`, double `fb`, double `fc`, double `xtol`, `gsl_vector` \*`x1x`, `gsl_vector` \*`dx1x`, `gsl_vector` \*`x2x`, `gsl_vector` \*`dx2x`, `gsl_vector` \*`gradient`, double \*`xstep`, double \*`f`, double \*`gnorm_u`) [`inline`, `protected`]

Perform the minimization.

Starting at  $(x_0, f_0)$  move along the direction  $p$  to find a minimum  $f(x_0 - \lambda p)$ , returning the new point  $x_1 = x_0 - \lambda p$ ,  $f_1 = f(x_1)$  and  $g_1 = \text{grad}(f)$  at  $x_1$ .

Definition at line 140 of file `gsl_mmin_conf.h`.

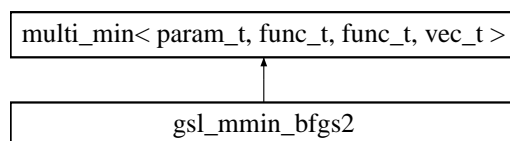
The documentation for this class was generated from the following file:

- `gsl_mmin_conf.h`

## 3.129 gsl\_mmin\_bfgs2 Class Template Reference

```
#include <gsl_mmin_bfgs2.h>
```

Inheritance diagram for `gsl_mmin_bfgs2`:



### 3.129.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc,
class dfunc_t = func_t> class gsl_mmin_bfgs2< param_t, func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t >
```

Multidimensional minimization by the BFGS algorithm (GSL).

This class includes the optimizations from the GSL minimizer `vector_bfgs2`.

Definition at line 373 of file `gsl_mmin_bfgs2.h`.

#### Public Member Functions

- `gsl_mmin_bfgs2()`
- virtual `~gsl_mmin_bfgs2()`
- virtual `int iterate()`  
*Perform an iteration.*
- virtual `const char * type()`  
*Return string denoting type("gsl\_mmin\_bfgs2").*
- virtual `int allocate(size_t n)`  
*Allocate the memory.*
- virtual `int free()`  
*Free the allocated memory.*
- `int restart()`  
*Reset the minimizer to use the current point as a new starting point.*
- virtual `int set(vec_t &x, double u_step_size, double tol_u, func_t &ufunc, param_t &upa)`  
*Set the function and initial guess.*
- virtual `int mmin(size_t nn, vec_t &xx, double &fmin, param_t &pa, func_t &ufunc)`  
*Calculate the minimum min of func w.r.t the array x of size nvar.*

#### Data Fields

- `double step_size`  
*The size of the first trial step.*
- `double lmin_tol`  
*The tolerance for the 1-dimensional minimizer.*

**Protected Attributes**

- `gsl_mmin_linmin lm`  
*The line minimizer.*
- `size_t dim`  
*Memory size.*
- `alloc_t ao`  
*Memory allocation.*

**The original variables from the GSL state structure**

- `int iter`
- `double step`
- `double g0norm`
- `double pnorm`
- `double delta_f`
- `double fp0`
- `gsl_vector * x0`
- `gsl_vector * g0`
- `gsl_vector * p`
- `gsl_vector * dx0`
- `gsl_vector * dg0`
- `gsl_mmin_wrapper< param_t, func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t > wrap`
- `double rho`
- `double sigma`
- `double tau1`
- `double tau2`
- `double tau3`
- `int order`

**Store the arguments to set() so we can use them for iterate()**

- `vec_t * st_x`
- `gsl_vector * st_dx`
- `gsl_vector * st_grad`
- `double st_f`

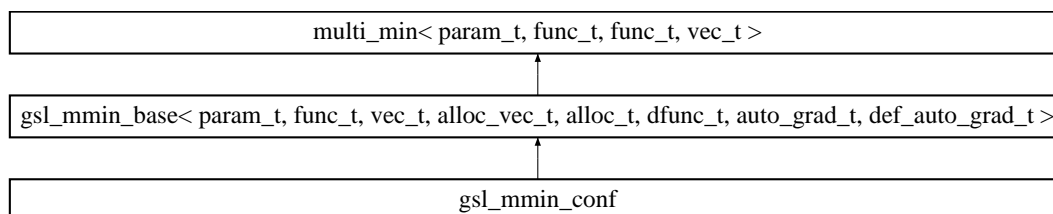
The documentation for this class was generated from the following file:

- `gsl_mmin_bfgs2.h`

**3.130 gsl\_mmin\_conf Class Template Reference**

```
#include <gsl_mmin_conf.h>
```

Inheritance diagram for `gsl_mmin_conf`:



### 3.130.1 Detailed Description

```
template<class param_t, class func_t = multi_func_t<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc, class dfunc_t = grad_func_t<param_t, ovector_view>, class auto_grad_t = gradient<param_t, func_t,
ovector_view>, class def_auto_grad_t = simple_grad<param_t, func_t, ovector_view>> class gsl_mmin_conf< param_t,
func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >
```

Multidimensional minimization by the Fletcher-Reeves conjugate [gradient](#) algorithm (GSL).

The variable [multi\\_min::tolf](#) is used as the maximum value of the [gradient](#) and is  $10^{-4}$  by default.

The [gsl\\_iterate\(\)](#) function for this minimizer chooses to return `GSL_ENOPROG` if the iteration fails to make progress without calling the error handler. This is presumably because the [iterate\(\)](#) function can fail to make progress when the algorithm has succeeded in finding the minimum. I prefer to return a non-zero value from a function only in cases where the error handler will also be called, so the user is clear on what an "error" means in the local context. Thus if [iterate\(\)](#) is failing to make progress, instead of returning a non-zero value, it sets the value of [it\\_info](#) to a non-zero value.

#### Todo

A bit of needless copying is required in the function wrapper to convert from `gsl_vector` to the templated vector type. This can be fixed, probably by rewriting `take_step` to produce a `vec_t &x1` rather than a `gsl_vector *x1`;

Those who look in detail at the code will note that the state variable `max_iter` has not been included here, because it was not really used in the original GSL code for these minimizers.

Definition at line 362 of file `gsl_mmin_conf.h`.

#### Public Member Functions

- [gsl\\_mmin\\_conf](#) ()
- virtual [~gsl\\_mmin\\_conf](#) ()
- virtual int [iterate](#) ()  
*Perform an iteration.*
- virtual const char \* [type](#) ()  
*Return string denoting type("gsl\_mmin\_conf").*
- virtual int [allocate](#) (size\_t n)  
*Allocate the memory.*
- virtual int [free](#) ()  
*Free the allocated memory.*
- int [restart](#) ()  
*Reset the minimizer to use the current point as a new starting point.*
- virtual int [set](#) (vec\_t &x, double u\_step\_size, double tol\_u, func\_t &ufunc, param\_t &pa)  
*Set the function and initial guess.*
- virtual int [set\\_de](#) (vec\_t &x, double u\_step\_size, double tol\_u, func\_t &ufunc, dfunc\_t &udfunc, param\_t &pa)  
*Set the function and initial guess.*
- virtual int [mmin](#) (size\_t nn, vec\_t &xx, double &fmin, param\_t &pa, func\_t &ufunc)  
*Calculate the minimum min of func w.r.t the array x of size nvar.*
- virtual int [mmin\\_de](#) (size\_t nn, vec\_t &xx, double &fmin, param\_t &pa, func\_t &ufunc, dfunc\_t &udfunc)  
*Calculate the minimum min of func w.r.t the array x of size nvar.*

#### Data Fields

- double [lmin\\_tol](#)  
*Tolerance for the line minimization (default  $10^{-4}$ ).*
- double [step\\_size](#)  
*Size of the initial step.*
- int [it\\_info](#)  
*Information from the last call to [iterate\(\)](#).*

## Protected Attributes

- alloc\_vec\_t [avt5](#)  
*Temporary vector.*
- alloc\_vec\_t [avt6](#)  
*Temporary vector.*
- alloc\_vec\_t [avt7](#)  
*Temporary vector.*
- alloc\_vec\_t [avt8](#)  
*Temporary vector.*

## The original variables from the GSL state structure

- int [iter](#)  
*Desc.*
- double [step](#)  
*Desc.*
- double [tol](#)  
*Desc.*
- gsl\_vector \* [x1](#)  
*Desc.*
- gsl\_vector \* [dx1](#)  
*Desc.*
- gsl\_vector \* [x2](#)  
*Desc.*
- double [pnorm](#)  
*Desc.*
- gsl\_vector \* [p](#)  
*Desc.*
- double [g0norm](#)  
*Desc.*
- gsl\_vector \* [g0](#)  
*Desc.*

## Store the arguments to set() so we can use them for iterate()

- gsl\_vector \* [ugx](#)  
*Desc.*
- gsl\_vector \* [ugg](#)  
*Desc.*
- gsl\_vector \* [udx](#)  
*Desc.*
- double [it\\_min](#)  
*Desc.*

## 3.130.2 Member Function Documentation

**3.130.2.1** `virtual int set (vec_t & x, double u_step_size, double tol_u, func_t & ufunc, param_t & pa)` [inline, virtual]

Set the function and initial guess.

Evaluate the function and its [gradient](#)

Definition at line 663 of file `gsl_mmin_conf.h`.

**3.130.2.2** `virtual int set_de (vec_t & x, double u_step_size, double tol_u, func_t & ufunc, dfunc_t & udfunc, param_t & pa)` [inline, virtual]

Set the function and initial guess.

Evaluate the function and its [gradient](#)

Definition at line 699 of file `gsl_mmin_conf.h`.



### 3.130.3 Field Documentation

#### 3.130.3.1 int it\_info

Information from the last call to [iterate\(\)](#).

This is 1 if `pnorm` or `gnorm` are 0 and 2 if `stepb` is zero.

#### Todo

Document this better

Definition at line 442 of file `gsl_mmin_conf.h`.

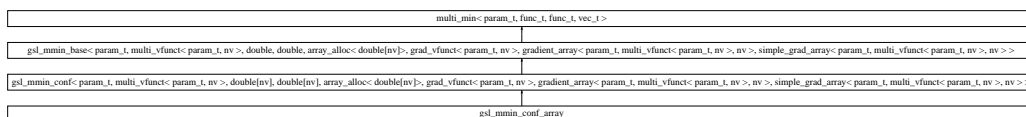
The documentation for this class was generated from the following file:

- `gsl_mmin_conf.h`

## 3.131 gsl\_mmin\_conf\_array Class Template Reference

```
#include <gsl_mmin_conf.h>
```

Inheritance diagram for `gsl_mmin_conf_array`::



### 3.131.1 Detailed Description

```
template<class param_t, size_t nv> class gsl_mmin_conf_array< param_t, nv >
```

An array version of [gsl\\_mmin\\_conf](#).

Definition at line 830 of file `gsl_mmin_conf.h`.

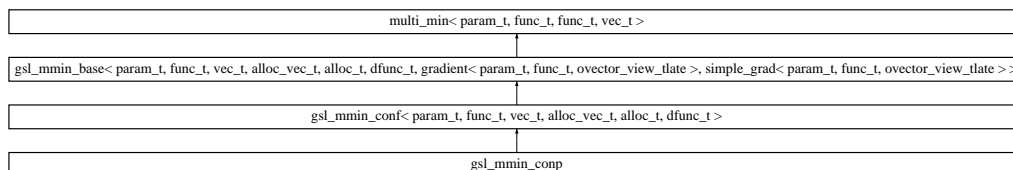
The documentation for this class was generated from the following file:

- `gsl_mmin_conf.h`

## 3.132 gsl\_mmin\_conp Class Template Reference

```
#include <gsl_mmin_conp.h>
```

Inheritance diagram for `gsl_mmin_conp`::



### 3.132.1 Detailed Description

```
template<class param_t, class func_t = multi_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc, class dfunc_t = grad_funct<param_t,vec_t>> class gsl_mmin_conp< param_t, func_t, vec_t, alloc_
vec_t, alloc_t, dfunc_t >
```

Multidimensional minimization by the Polak-Ribiere conjugate [gradient](#) algorithm (GSL).

The variable [multi\\_min::tolf](#) is used as the maximum value of the [gradient](#) and is  $10^{-4}$  by default.

The `gsl_iterate()` function for this minimizer chooses to return `GSL_ENOPROG` if the iteration fails to make progress without calling the error handler. This is presumably because the `iterate()` function can fail to make progress when the algorithm has succeeded in finding the minimum. I prefer to return a non-zero value from a function only in cases where the error handler will also be called, so the user is clear on what an "error" means in the local context. Thus if `iterate()` is failing to make progress, instead of returning a non-zero value, it sets the value of `it_info` to a non-zero value.

#### Todo

A bit of needless copying is required in the function wrapper to convert from `gsl_vector` to the templated vector type. This can be fixed.

#### Todo

Document stopping conditions

Definition at line 60 of file `gsl_mmin_conp.h`.

### Public Member Functions

- virtual int [iterate](#) ()  
*Perform an iteration.*
- virtual const char \* [type](#) ()  
*Return string denoting type("gsl\_mmin\_conp").*

The documentation for this class was generated from the following file:

- `gsl_mmin_conp.h`

## 3.133 gsl\_mmin\_linmin Class Reference

```
#include <gsl_mmin_bfgs2.h>
```

### 3.133.1 Detailed Description

The line minimizer for [gsl\\_mmin\\_bfgs2](#).

Definition at line 307 of file `gsl_mmin_bfgs2.h`.

### Public Member Functions

- int [minimize](#) ([gsl\\_mmin\\_wrap\\_base](#) &wrap, double rho, double sigma, double tau1, double tau2, double tau3, int order, double alpha1, double \*alpha\_new)  
*The line minimization.*

## Protected Member Functions

- double [interp\\_quad](#) (double f0, double fp0, double f1, double zl, double zh)  
*Minimize the interpolating quadratic.*
- double [cubic](#) (double c0, double c1, double c2, double c3, double z)  
*Minimize the interpolating cubic.*
- void [check\\_extremum](#) (double c0, double c1, double c2, double c3, double z, double \*zmin, double \*fmin)  
*Test to see curvature is positive.*
- double [interp\\_cubic](#) (double f0, double fp0, double f1, double fp1, double zl, double zh)  
*Interpolate using a cubic.*
- double [interpolate](#) (double a, double fa, double fpa, double b, double fb, double fpb, double xmin, double xmax, int order)  
*Perform the interpolation.*

### 3.133.2 Member Function Documentation

#### 3.133.2.1 double interp\_quad (double f0, double fp0, double f1, double zl, double zh) [protected]

Minimize the interpolating quadratic.

Find a minimum in  $x=[0,1]$  of the interpolating quadratic through  $(0,f_0)$   $(1,f_1)$  with derivative  $fp_0$  at  $x=0$ . The interpolating polynomial is  $q(x) = f_0 + fp_0 * x + (f_1 - f_0 - fp_0) * x^2$

#### 3.133.2.2 double cubic (double c0, double c1, double c2, double c3, double z) [protected]

Minimize the interpolating cubic.

Find a minimum in  $x=[0,1]$  of the interpolating cubic through  $(0,f_0)$   $(1,f_1)$  with derivatives  $fp_0$  at  $x=0$  and  $fp_1$  at  $x=1$ .

The interpolating polynomial is:

$$c(x) = f_0 + fp_0 * x + \eta * x^2 + \xi * x^3$$

where  $\eta = 3 * (f_1 - f_0) - 2 * fp_0 - fp_1$ ,  $\xi = fp_0 + fp_1 - 2 * (f_1 - f_0)$ .

#### 3.133.2.3 int minimize (gsl\_mmin\_wrap\_base & wrap, double rho, double sigma, double tau1, double tau2, double tau3, int order, double alpha1, double \* alpha\_new)

The line minimization.

recommended values from Fletcher are  $\rho = 0.01$ ,  $\sigma = 0.1$ ,  $\tau_1 = 9$ ,  $\tau_2 = 0.05$ ,  $\tau_3 = 0.5$

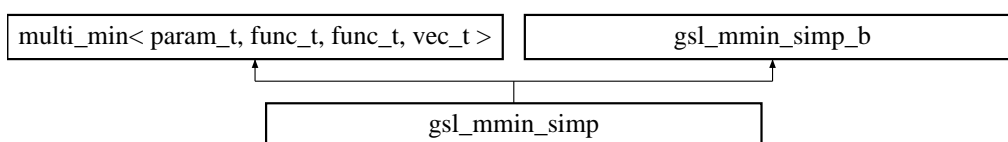
The documentation for this class was generated from the following file:

- `gsl_mmin_bfgs2.h`

## 3.134 gsl\_mmin\_simp Class Template Reference

```
#include <gsl_mmin_simp.h>
```

Inheritance diagram for `gsl_mmin_simp`:



### 3.134.1 Detailed Description

```
template<class param_t, class func_t = multi_func_t<param_t>, class vec_t = ovector_view> class gsl_mmin_simp<
param_t, func_t, vec_t >
```

Multidimensional minimization by the Simplex method (GSL).

#### Todo

Not properly generalized to non-GSL vectors (to do this, I'll have to store the simplex as a `vec_t` array rather than a `gsl_matrix`). Right now, this only works with `vec_t = ovector_view`.

#### Todo

Add a `minimize` function which allows specification of the entire simplex.

#### Todo

Gracefully ensure memory allocation and deallocation is performed automatically.

#### Todo

Test `mmin_twovec()`.

#### Todo

Document how `set()` chooses the simplex from the initial guess and step size

Definition at line 49 of file `gsl_mmin_simp.h`.

### Public Member Functions

- `gsl_mmin_simp()`
- virtual `~gsl_mmin_simp()`
- virtual int `mmin` (size\_t nn, vec\_t &xx, double &fmin, param\_t &pa, func\_t &ufunc)  
*Calculate the minimum min of func w.r.t the array x of size nvar.*
- virtual int `mmin_twovec` (size\_t nn, vec\_t &xx, vec\_t &xx2, double &fmin, param\_t &pa, func\_t &ufunc)  
*Calculate the minimum min of func w.r.t the array x of size nvar, using xx and xx2 to specify the simplex.*
- virtual int `allocate` (size\_t n)  
*Allocate the memory.*
- virtual int `free` ()  
*Free the allocated memory.*
- virtual int `set` (func\_t &ufunc, param\_t &pa, vec\_t &ax, vec\_t &step\_size)  
*Set the function and initial guess.*
- virtual int `iterate` ()  
*Perform an iteration.*
- virtual int `print_iter` (size\_t nv, vec\_t &xx, double y, int iter, double value, double limit, std::string comment)  
*Print out iteration information.*
- virtual const char \* `type` ()  
*Return string denoting type("gsl\_mmin\_simp").*

### Data Fields

- double `initial_step`  
*The initial stepsize (default 1.0).*
- int `print_simplex`  
*Print simplex information in `print_iter()` (default 0).*

## Protected Member Functions

- virtual double [move\\_corner](#) (const double coeff, const simp\_state\_t \*state, size\_t corner, gsl\_vector \*xc, func\_t &f, size\_t nvar, param\_t &pa)  
*Move a corner of a simplex.*
- virtual int [contract\\_by\\_best](#) (simp\_state\_t \*state, size\_t best, gsl\_vector \*xc, func\_t &f, size\_t nvar, param\_t &pa)  
*Contract the simplex towards the best point.*

## Protected Attributes

- size\_t [dim](#)  
*Number of variables to be minimized over.*
- gsl\_vector \* [x](#)  
*Present minimum vector.*
- double [fval](#)  
*Function value at minimum.*
- func\_t \* [func](#)  
*Function.*
- param\_t \* [params](#)  
*Parameters.*
- bool [set\\_called](#)  
*True if [set\(\)](#) has been called.*
- double [size](#)  
*Size of simplex.*
- simp\_state\_t [lstate](#)  
*Simplex state storage.*

### 3.134.2 Member Function Documentation

**3.134.2.1 virtual double move\_corner (const double coeff, const simp\_state\_t \* state, size\_t corner, gsl\_vector \* xc, func\_t & f, size\_t nvar, param\_t & pa)** [inline, protected, virtual]

Move a corner of a simplex.

Moves a simplex corner scaled by coeff (negative value represents mirroring by the middle point of the "other" corner points) and gives new corner in xc and function value at xc as a return value.

Definition at line 65 of file `gsl_mmin_simp.h`.

**3.134.2.2 virtual int contract\_by\_best (simp\_state\_t \* state, size\_t best, gsl\_vector \* xc, func\_t & f, size\_t nvar, param\_t & pa)** [inline, protected, virtual]

Contract the simplex towards the best point.

Function contracts the simplex in respect to best valued corner. All corners besides the best corner are moved.

The xc vector is used as work space.

Definition at line 105 of file `gsl_mmin_simp.h`.

**3.134.2.3 virtual int print\_iter (size\_t nv, vec\_t & xx, double y, int iter, double value, double limit, std::string comment)** [inline, virtual]

Print out iteration information.

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to `std::cout` along with the iteration number. If verbose>=2 then each iteration waits for a character.

Definition at line 513 of file `gsl_mmin_simp.h`.

### 3.134.3 Field Documentation

#### 3.134.3.1 int print\_simplex

Print simplex information in [print\\_iter\(\)](#) (default 0).

If this is 1 and [verbose](#) is greater than 0, then [print\\_iter\(\)](#) will print the function values at all the simplex points.

Definition at line 179 of file `gsl_mmin_simp.h`.

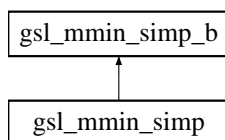
The documentation for this class was generated from the following file:

- `gsl_mmin_simp.h`

## 3.135 gsl\_mmin\_simp\_b Class Reference

```
#include <gsl_mmin_simp_b.h>
```

Inheritance diagram for `gsl_mmin_simp_b`:



### 3.135.1 Detailed Description

Base routines for the GSL simplex minimizer.

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

#### Public Member Functions

- int [nmsimplex\\_calc\\_center](#) (const [simp\\_state\\_t](#) \*state, [gsl\\_vector](#) \*mp)  
*Compute the center of the simplex and store in mp.*
- double [nmsimplex\\_size](#) ([simp\\_state\\_t](#) \*state)  
*Compute the size of the simplex.*

#### Data Structures

- struct [simp\\_state\\_t](#)  
*State type for GSL simplex minimizer.*

### 3.135.2 Member Function Documentation

#### 3.135.2.1 double nmsimplex\_size (simp\_state\_t \* state)

Compute the size of the simplex.

Calculates simplex size as average sum of length of vectors from simplex center to corner points:

$$(1/n) \sum ||y - y_{\text{middlepoint}}||$$

The documentation for this class was generated from the following file:

- `gsl_mmin_simp_b.h`

## 3.136 gsl\_mmin\_simp\_b::simp\_state\_t Struct Reference

```
#include <gsl_mmin_simp_b.h>
```

### 3.136.1 Detailed Description

State type for GSL simplex minimizer.

Definition at line 43 of file `gsl_mmin_simp_b.h`.

#### Data Fields

- `gsl_matrix * x1`  
*The  $(n+1,n)$  matrix containing the simplex.*
- `gsl_vector * y1`  
*The  $(n+1)$  function values at the simplex points.*
- `gsl_vector * ws1`  
*Desc.*
- `gsl_vector * ws2`  
*Desc.*

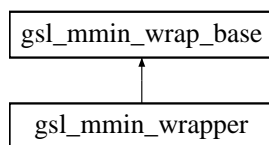
The documentation for this struct was generated from the following file:

- `gsl_mmin_simp_b.h`

## 3.137 gsl\_mmin\_wrap\_base Class Reference

```
#include <gsl_mmin_bfgs2.h>
```

Inheritance diagram for `gsl_mmin_wrap_base`:



### 3.137.1 Detailed Description

Virtual base for the `gsl_mmin_bfgs2` wrapper.

This is useful so that the `gsl_mmin_linmin` class doesn't need to depend on any template parameters, even though it will need a wrapping object as an argument for the `gsl_mmin_linmin::minimize()` function.

Definition at line 43 of file `gsl_mmin_bfgs2.h`.

#### Public Member Functions

- virtual `~gsl_mmin_wrap_base()`
- virtual double `wrap_f` (double alpha, void \*params)=0  
*Function.*
- virtual double `wrap_df` (double alpha, void \*params)=0  
*Derivative.*

- virtual void [wrap\\_fdf](#) (double alpha, void \*params, double \*f, double \*df)=0  
*Function and derivative.*

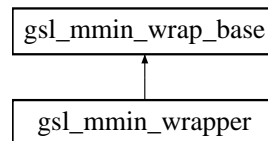
The documentation for this class was generated from the following file:

- [gsl\\_mmin\\_bfgs2.h](#)

### 3.138 gsl\_mmin\_wrapper Class Template Reference

```
#include <gsl_mmin_bfgs2.h>
```

Inheritance diagram for `gsl_mmin_wrapper`:



#### 3.138.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t = ovector\_view, class alloc\_vec\_t = ovector, class alloc\_t = ovector\_alloc, class dfunc\_t = func\_t> class `gsl_mmin_wrapper`< param\_t, func\_t, vec\_t, alloc\_vec\_t, alloc\_t, dfunc\_t >**

Wrapper class for the [gsl\\_mmin\\_bfgs2](#) minimizer.

#### Idea for future

There's a bit of extra vector copying here which could potentially be avoided.

Definition at line 63 of file [gsl\\_mmin\\_bfgs2.h](#).

#### Public Member Functions

- void [prepare\\_wrapper](#) (func\_t &ufunc, param\_t &upa, gsl\_vector \*t\_x, double f, gsl\_vector \*t\_g, gsl\_vector \*t\_p)  
*Initialize wrapper.*
- void [update\\_position](#) (double alpha, gsl\_vector \*t\_x, double \*t\_f, gsl\_vector \*t\_g)  
*Update position.*
- void [change\\_direction](#) ()  
*Convert cache values to the new minimizer direction.*

#### Data Fields

- alloc\_vec\_t [av\\_x\\_alpha](#)  
*Temporary storage.*
- alloc\_vec\_t [av\\_g\\_alpha](#)  
*Temporary storage.*
- size\_t [dim](#)  
*Number of minimization dimensions.*



**Protected Member Functions**

- void [moveto](#) (double alpha)  
*Move to a new point, using the cached value if possible.*
- double [slope](#) ()  
*Compute the slope.*
- virtual double [wrap\\_f](#) (double alpha, void \*params)  
*Evaluate the function.*
- virtual double [wrap\\_df](#) (double alpha, void \*params)  
*Evaluate the derivative.*
- int [simple\\_df](#) (vec\_t &x2, vec\_t &g2)  
*A simple derivative.*
- virtual void [wrap\\_fdf](#) (double alpha, void \*params, double \*f, double \*df)  
*Evaluate the function and the derivative.*

**Protected Attributes**

- func\_t \* [func](#)  
*Function.*
- dfunc\_t \* [dfunc](#)  
*Derivative.*
- param\_t \* [pa](#)  
*Parameters.*

**fixed values**

- gsl\_vector \* [x](#)
- gsl\_vector \* [g](#)
- gsl\_vector \* [p](#)

**cached values, for  $x(\alpha) = x + \alpha * p$** 

- double [f\\_alpha](#)
- double [df\\_alpha](#)

**cache "keys"**

- double [f\\_cache\\_key](#)
- double [df\\_cache\\_key](#)
- double [x\\_cache\\_key](#)
- double [g\\_cache\\_key](#)

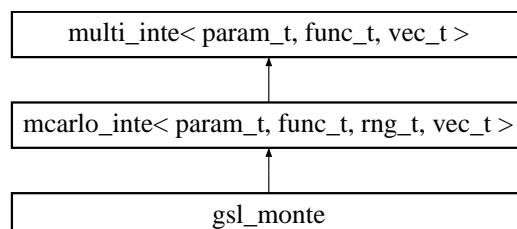
The documentation for this class was generated from the following file:

- [gsl\\_mmin\\_bfgs2.h](#)

**3.139 gsl\_monte Class Template Reference**

```
#include <gsl_monte.h>
```

Inheritance diagram for `gsl_monte`:



### 3.139.1 Detailed Description

`template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_view> class gsl_monte< param_t, func_t, rng_t, vec_t >`

Multidimensional integration using plain Monte Carlo (GSL).

Definition at line 42 of file `gsl_monte.h`.

#### Public Member Functions

- virtual `~gsl_monte()`
- virtual `int minteg_err(func_t &func, size_t ndim, const vec_t &a, const vec_t &b, param_t &pa, double &res, double &err)`  
Integrate function `func` from  $x=a$  to  $x=b$ .
- virtual `const char * type()`  
Return string denoting type ("`gsl_monte`").

#### Static Protected Member Functions

- static `double gsl_func(double *x, size_t dim, void *pa)`  
The GSL function pointer.

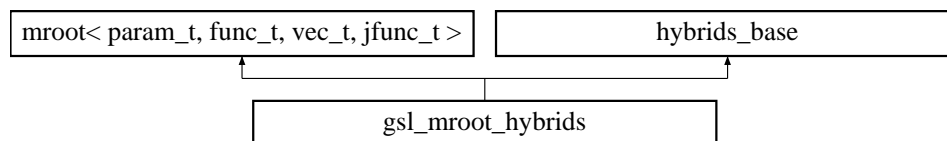
The documentation for this class was generated from the following file:

- `gsl_monte.h`

## 3.140 gsl\_mroot\_hybrids Class Template Reference

`#include <gsl_mroot_hybrids.h>`

Inheritance diagram for `gsl_mroot_hybrids`:



### 3.140.1 Detailed Description

`template<class param_t, class func_t = mm_func<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class jfunc_t = jac_func<param_t,vec_t,omatrix_view>> class gsl_mroot_hybrids< param_t, func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t >`

Multidimensional root-finding algorithm using Powell's Hybrid method (GSL).

This is a recasted version of the GSL routines which use a modified version of Powell's Hybrid method as implemented in the HYBRJ algorithm in MINPACK. Both the scaled and unscaled options are available by setting `int_scaling` (the scaled version is the default). If derivatives are not provided, they will be computed automatically. This class provides the GSL-like interface using `allocate()`, `set()` (or `set_de()` in case where derivatives are available), `iterate()`, and `free()` and higher-level interfaces, `msolve()` and `msolve_de()`, which perform the solution automatically. Some additional checking is performed in case the user calls the functions out of order (i.e. `set()` without `allocate()`).

The functions `msolve()` and `msolve_de()` use the condition  $\sum_i |f_i| < \text{mroot::tolf}$  to determine if the solver has succeeded.

The original GSL algorithm has been modified to shrink the stepsize if a proposed step causes the function to return a non-zero value. This allows the routine to automatically try to avoid regions where the function is not defined. To return to the default GSL behavior, set [shrink\\_step](#) to false.

## Todo

Internally, there is a little unnecessary copying back and forth of vectors

Definition at line 72 of file `gsl_mroot_hybrids.h`.

## Public Member Functions

- [gsl\\_mroot\\_hybrids](#) ()
- virtual [~gsl\\_mroot\\_hybrids](#) ()
- virtual int [set\\_jacobian](#) ([jacobian](#)< param\_t, func\_t, vec\_t, [omatrix\\_view](#) > &j)  
*Set the automatic Jacobian object.*
- int [iterate](#) (vec\_t &ux)  
*Perform an iteration.*
- int [allocate](#) (size\_t n)  
*Allocate the memory.*
- int [free](#) ()  
*Free the allocated memory.*
- virtual const char \* [type](#) ()  
*Return the type, "gsl\_mroot\_hybrids".*
- virtual int [msolve\\_de](#) (size\_t nn, vec\_t &xx, param\_t &pa, func\_t &ufunc, jfunc\_t &dfunc)  
*Solve func with derivatives dfunc using x as an initial guess, returning x.*
- virtual int [msolve](#) (size\_t nn, vec\_t &xx, param\_t &pa, func\_t &ufunc)  
*Solve ufunc using xx as an initial guess, returning xx.*
- int [set](#) (size\_t nn, vec\_t &ax, func\_t &ufunc, param\_t &pa)  
*Set the function, the parameters, and the initial guess.*
- int [set\\_de](#) (size\_t nn, vec\_t &ax, func\_t &ufunc, jfunc\_t &dfunc, param\_t &pa)  
*Set the function, the Jacobian, the parameters, and the initial guess.*

## Data Fields

- bool [shrink\\_step](#)  
*If true, [iterate\(\)](#) will shrink the step-size automatically if the function returns a non-zero value (default true).*
- bool [int\\_scaling](#)  
*If true, use the internal scaling method (default true).*
- [simple\\_jacobian](#)< param\_t, func\_t, vec\_t, [omatrix\\_view](#), alloc\_vec\_t, alloc\_t > [def\\_jac](#)  
*Default automatic Jacobian object.*
- [gsl\\_vector](#) \* [f](#)  
*The value of the function at the present iteration.*

## Protected Member Functions

- int [solve\\_set](#) (size\_t nn, vec\_t &xx, param\_t &pa, func\_t &ufunc)  
*Finish the solution after [set\(\)](#) or [set\\_de\(\)](#) has been called.*

## Protected Attributes

- jfunc\_t \* [jac](#)  
*Pointer to the user-specified Jacobian object.*
- [jacobian](#)< param\_t, func\_t, vec\_t, [omatrix\\_view](#) > \* [ajac](#)  
*Pointer to the automatic Jacobian object.*

- `alloc_t ao`  
*Memory allocator for objects of type `alloc_vec_t`.*
- `gsl_vector * x`  
*The present solution.*
- `gsl_vector * dx`  
*The value of the derivative.*
- `o2scl_hybrid_state_t * state`  
*The solver state.*
- `param_t * params`  
*The function parameters.*
- `size_t dim`  
*The number of equations and unknowns.*
- `bool jac_given`  
*True if the *jacobian* has been given.*
- `func_t * fnewp`  
*Pointer to the user-specified function.*
- `bool set_called`  
*True if "set" has been called.*

## Data Structures

- `struct o2scl_hybrid_state_t`  
*A structure for *gsl\_mroot\_hybrids*.*

### 3.140.2 Member Function Documentation

#### 3.140.2.1 `int iterate (vec_t & ux)` [inline]

Perform an iteration.

At the end of the iteration, the current value of the solution is stored in `ux`.

Definition at line 246 of file `gsl_mroot_hybrids.h`.

#### 3.140.2.2 `int set_de (size_t nn, vec_t & ax, func_t & ufunc, jfunc_t & dfunc, param_t & pa)` [inline]

Set the function, the Jacobian, the parameters, and the initial guess.

Make sure `set()` uses the right Jacobian

Reset `jac_given` since `set()` will set it back to false

Definition at line 694 of file `gsl_mroot_hybrids.h`.

### 3.140.3 Field Documentation

#### 3.140.3.1 `bool shrink_step`

If true, `iterate()` will shrink the step-size automatically if the function returns a non-zero value (default true).

The original GSL behavior can be obtained by setting this to `false`.

Definition at line 214 of file `gsl_mroot_hybrids.h`.

The documentation for this class was generated from the following file:

- `gsl_mroot_hybrids.h`

## 3.141 gsl\_mroot\_hybrids::o2scl\_hybrid\_state\_t Struct Reference

```
#include <gsl_mroot_hybrids.h>
```

### 3.141.1 Detailed Description

```
template<class param_t, class func_t = mm_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc, class jfunc_t = jac_funct<param_t,vec_t,omatrix_view>> struct gsl_mroot_hybrids< param_t,
func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t >::o2scl_hybrid_state_t
```

A structure for [gsl\\_mroot\\_hybrids](#).

Definition at line 89 of file [gsl\\_mroot\\_hybrids.h](#).

### Data Fields

- size\_t [iter](#)
- size\_t [ncfail](#)
- size\_t [ncsuc](#)
- size\_t [nslow1](#)
- size\_t [nslow2](#)
- double [fnorm](#)
- double [delta](#)
- gsl\_matrix \* [J](#)
- gsl\_matrix \* [q](#)
- gsl\_matrix \* [r](#)
- gsl\_vector \* [tau](#)
- gsl\_vector \* [diag](#)
- gsl\_vector \* [qtf](#)
- gsl\_vector \* [newton](#)
- gsl\_vector \* [gradient](#)
- gsl\_vector \* [x\\_trial](#)
- gsl\_vector \* [f\\_trial](#)
- gsl\_vector \* [df](#)
- gsl\_vector \* [qtdf](#)
- gsl\_vector \* [rdx](#)
- gsl\_vector \* [w](#)
- gsl\_vector \* [v](#)

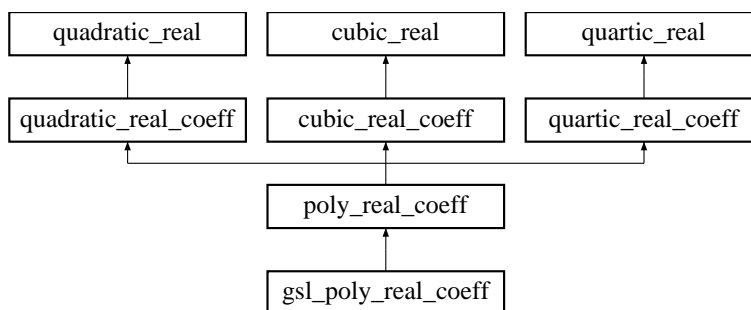
The documentation for this struct was generated from the following file:

- [gsl\\_mroot\\_hybrids.h](#)

## 3.142 gsl\_poly\_real\_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for [gsl\\_poly\\_real\\_coeff](#):



### 3.142.1 Detailed Description

Solve a general polynomial with real coefficients (GSL).

Definition at line 538 of file poly.h.

#### Public Member Functions

- virtual int [solve\\_rc](#) (int n, const double co[ ], std::complex< double > ro[ ])
 

*Solve the n-th order polynomial.*
- virtual int [solve\\_rc](#) (const double a3, const double b3, const double c3, const double d3, double &x1, std::complex< double > &x2, std::complex< double > &x3)
 

*Solves the polynomial  $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$  giving the real solution  $x = x_1$  and two complex solutions  $x = x_1$ ,  $x = x_2$ , and  $x = x_3$ .*
- virtual int [solve\\_rc](#) (const double a2, const double b2, const double c2, std::complex< double > &x1, std::complex< double > &x2)
 

*Solves the polynomial  $a_2x^2 + b_2x + c_2 = 0$  giving the two complex solutions  $x = x_1$  and  $x = x_2$ .*
- virtual int [solve\\_rc](#) (const double a4, const double b4, const double c4, const double d4, const double e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)
 

*Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four complex solutions  $x = x_1, x_2, x_3, x_4$ .*
- const char \* [type](#) ()
 

*Return a string denoting the type ("gsl\_poly\_real\_coeff").*

#### Protected Attributes

- gsl\_poly\_complex\_workspace \* [w2](#)
- gsl\_poly\_complex\_workspace \* [w3](#)
- gsl\_poly\_complex\_workspace \* [w4](#)
- gsl\_poly\_complex\_workspace \* [wgen](#)
- int [gen\\_size](#)

*The size of the workspace [wgen](#).*

### 3.142.2 Member Function Documentation

#### 3.142.2.1 virtual int solve\_rc (int n, const double co[ ], std::complex< double > ro[ ]) [virtual]

Solve the n-th order polynomial.

The coefficients are stored in co[], with the leading coefficient as co[0] and the constant term as co[n]. The roots are returned in ro[0],...,ro[n-1].

Reimplemented from [poly\\_real\\_coeff](#).

**3.142.2.2 virtual int solve\_rc** (const double *a4*, const double *b4*, const double *c4*, const double *d4*, const double *e4*, std::complex< double > & *x1*, std::complex< double > & *x2*, std::complex< double > & *x3*, std::complex< double > & *x4*) [virtual]

Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four complex solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented from [quartic\\_real\\_coeff](#).

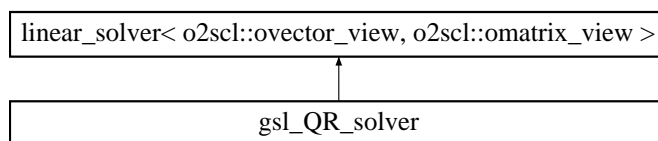
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.143 gsl\_QR\_solver Class Reference

```
#include <ode_it_solve.h>
```

Inheritance diagram for `gsl_QR_solver`:



### 3.143.1 Detailed Description

GSL solver by QR decomposition.

Definition at line 156 of file `ode_it_solve.h`.

#### Public Member Functions

- virtual int [solve](#) (size\_t *n*, o2scl::omatrix\_view &*A*, o2scl::ovector\_view &*b*, o2scl::ovector\_view &*x*)  
Solve square linear system  $Ax = b$  of size *n*.
- virtual [~gsl\\_QR\\_solver](#) ()

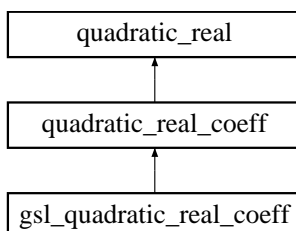
The documentation for this class was generated from the following file:

- `ode_it_solve.h`

## 3.144 gsl\_quadratic\_real\_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for `gsl_quadratic_real_coeff`:



### 3.144.1 Detailed Description

Solve a quadratic with real coefficients and complex roots (GSL).

Definition at line 443 of file poly.h.

#### Public Member Functions

- virtual `~gsl_quadratic_real_coeff()`
- virtual int `solve_rc` (const double a2, const double b2, const double c2, std::complex< double > &x1, std::complex< double > &x2)  
*Solves the polynomial  $a_2x^2 + b_2x + c_2 = 0$  giving the two complex solutions  $x = x_1$  and  $x = x_2$ .*
- const char \* `type()`  
*Return a string denoting the type ("gsl\_quadratic\_real\_coeff").*

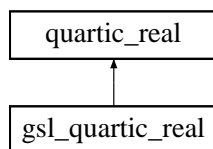
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.145 gsl\_quartic\_real Class Reference

```
#include <poly.h>
```

Inheritance diagram for `gsl_quartic_real`:



### 3.145.1 Detailed Description

Solve a quartic with real coefficients and real roots (GSL).

Definition at line 498 of file poly.h.

#### Public Member Functions

- virtual `~gsl_quartic_real()`
- virtual int `solve_r` (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)  
*Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .*
- const char \* `type()`  
*Return a string denoting the type ("gsl\_quartic\_real").*

### 3.145.2 Member Function Documentation

**3.145.2.1** virtual int `solve_r` (const double *a4*, const double *b4*, const double *c4*, const double *d4*, const double *e4*, double &*x1*, double &*x2*, double &*x3*, double &*x4*) [virtual]

Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented from [quartic\\_real](#).

The documentation for this class was generated from the following file:

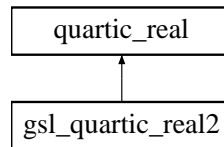


- [poly.h](#)

## 3.146 gsl\_quartic\_real2 Class Reference

```
#include <poly.h>
```

Inheritance diagram for `gsl_quartic_real2`:



### 3.146.1 Detailed Description

Solve a quartic with real coefficients and real roots (GSL).

#### Todo

Document the distinction between this class and [gsl\\_quartic\\_real](#)

Definition at line 520 of file `poly.h`.

### Public Member Functions

- virtual `~gsl_quartic_real2()`
- virtual `int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)`
- `const char * type()`  
*Return a string denoting the type ("gsl\_quartic\_real2").*

### 3.146.2 Member Function Documentation

**3.146.2.1** `virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)` [virtual]

Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented from [quartic\\_real](#).

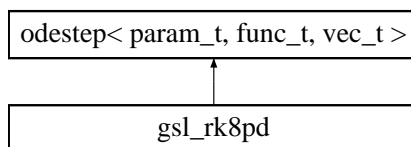
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.147 gsl\_rk8pd Class Template Reference

```
#include <gsl_rk8pd.h>
```

Inheritance diagram for `gsl_rk8pd`:



### 3.147.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t, class alloc\_vec\_t, class alloc\_t> class gsl\_rk8pd< param\_t, func\_t, vec\_t, alloc\_vec\_t, alloc\_t >**

Cash-Karp embedded Runge-Kutta formula (GSL).

Definition at line 37 of file `gsl_rk8pd.h`.

#### Public Member Functions

- [gsl\\_rk8pd](#) ()
- virtual [~gsl\\_rk8pd](#) ()
- virtual int [step](#) (double x, double h, size\_t n, vec\_t &y, vec\_t &dydx, vec\_t &yout, vec\_t &yerr, vec\_t &dydx\_out, param\_t &pa, func\_t &derivs)  
*Perform an integration step.*

#### Protected Attributes

- size\_t [ndim](#)  
*Size of allocated vectors.*
- alloc\_t [ao](#)  
*Memory allocator for objects of type alloc\_vec\_t.*

#### Storage for the intermediate steps

- alloc\_vec\_t [k2](#)
- alloc\_vec\_t [k3](#)
- alloc\_vec\_t [k4](#)
- alloc\_vec\_t [k5](#)
- alloc\_vec\_t [k6](#)
- alloc\_vec\_t [k7](#)
- alloc\_vec\_t [ytmp](#)
- alloc\_vec\_t [k8](#)
- alloc\_vec\_t [k9](#)
- alloc\_vec\_t [k10](#)
- alloc\_vec\_t [k11](#)
- alloc\_vec\_t [k12](#)
- alloc\_vec\_t [k13](#)

#### Storage for the coefficients

- double [Abar](#) [13]
- double [A](#) [12]
- double [ah](#) [10]
- double [b21](#)
- double [b3](#) [2]
- double [b4](#) [3]
- double [b5](#) [4]
- double [b6](#) [5]
- double [b7](#) [6]
- double [b8](#) [7]
- double [b9](#) [8]
- double [b10](#) [9]
- double [b11](#) [10]
- double [b12](#) [11]
- double [b13](#) [12]

### 3.147.2 Member Function Documentation

**3.147.2.1** `virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, param_t &pa, func_t &derivs)` [`inline`, `virtual`]

Perform an integration step.

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must generally be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

Reimplemented from [odestep](#).

Definition at line 227 of file `gsl_rk8pd.h`.

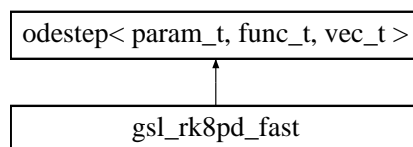
The documentation for this class was generated from the following file:

- `gsl_rk8pd.h`

## 3.148 gsl\_rk8pd\_fast Class Template Reference

```
#include <gsl_rk8pd.h>
```

Inheritance diagram for `gsl_rk8pd_fast`:



### 3.148.1 Detailed Description

`template<size_t N, class param_t, class func_t, class vec_t, class alloc_vec_t, class alloc_t> class gsl_rk8pd_fast< N, param_t, func_t, vec_t, alloc_vec_t, alloc_t >`

Cash-Karp embedded Runge-Kutta formula (GSL) without memory allocation and error checking.

This a fast version of [gsl\\_rk8pd](#), which is a stepper for a fixed number of ODEs. It ignores the error values returned by the `derivs` argument. The argument `n` to [step\(\)](#) must always be equal to the template parameter `N`, and the vector parameters to `step` must have space allocated for at least `N` elements. No error checking is performed to ensure that this is the case.

Definition at line 390 of file `gsl_rk8pd.h`.

#### Public Member Functions

- [gsl\\_rk8pd\\_fast](#) ()
- `virtual ~gsl_rk8pd_fast` ()
- `virtual int step` (double x, double h, size\_t n, vec\_t &y, vec\_t &dydx, vec\_t &yout, vec\_t &yerr, vec\_t &dydx\_out, param\_t &pa, func\_t &derivs)  
*Perform an integration step.*

#### Protected Attributes

- `alloc_t ao`

*Memory allocator for objects of type `alloc_vec_t`.*

### Storage for the intermediate steps

- `alloc_vec_t k2`
- `alloc_vec_t k3`
- `alloc_vec_t k4`
- `alloc_vec_t k5`
- `alloc_vec_t k6`
- `alloc_vec_t k7`
- `alloc_vec_t ytmp`
- `alloc_vec_t k8`
- `alloc_vec_t k9`
- `alloc_vec_t k10`
- `alloc_vec_t k11`
- `alloc_vec_t k12`
- `alloc_vec_t k13`

### Storage for the coefficients

- `double Abar [13]`
- `double A [12]`
- `double ah [10]`
- `double b21`
- `double b3 [2]`
- `double b4 [3]`
- `double b5 [4]`
- `double b6 [5]`
- `double b7 [6]`
- `double b8 [7]`
- `double b9 [8]`
- `double b10 [9]`
- `double b11 [10]`
- `double b12 [11]`
- `double b13 [12]`

## 3.148.2 Member Function Documentation

### 3.148.2.1 `virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, param_t &pa, func_t &derivs)` `[inline, virtual]`

Perform an integration step.

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must generally be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

#### Note:

The value of the parameter `n` must be equal to the template parameter `N`.

Reimplemented from [odestep](#).

Definition at line 592 of file `gsl_rk8pd.h`.

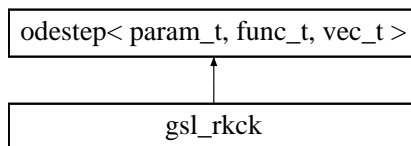
The documentation for this class was generated from the following file:

- `gsl_rk8pd.h`

## 3.149 gsl\_rkck Class Template Reference

```
#include <gsl_rkck.h>
```

Inheritance diagram for `gsl_rkck`:



### 3.149.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>
class gsl_rkck< param_t, func_t, vec_t, alloc_vec_t, alloc_t >
```

Cash-Karp embedded Runge-Kutta formula (GSL).

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

#### Public Member Functions

- [gsl\\_rkck](#) ()
- virtual [~gsl\\_rkck](#) ()
- virtual int [step](#) (double x, double h, size\_t n, vec\_t &y, vec\_t &dydx, vec\_t &yout, vec\_t &yerr, vec\_t &dydx\_out, param\_t &pa, func\_t &derivs)  
*Perform an integration step.*

#### Protected Attributes

- size\_t [ndim](#)  
*Size of allocated vectors.*
- alloc\_t [ao](#)  
*Memory allocator for objects of type `alloc_vec_t`.*

#### Storage for the intermediate steps

- alloc\_vec\_t [k2](#)
- alloc\_vec\_t [k3](#)
- alloc\_vec\_t [k4](#)
- alloc\_vec\_t [k5](#)
- alloc\_vec\_t [k6](#)
- alloc\_vec\_t [ytmp](#)

#### Storage for the coefficients

- double [ah](#) [5]
- double [b3](#) [2]
- double [b4](#) [3]
- double [b5](#) [4]
- double [b6](#) [5]
- double [ec](#) [7]
- double [b21](#)
- double [c1](#)
- double [c3](#)
- double [c4](#)
- double [c6](#)

### 3.149.2 Member Function Documentation

**3.149.2.1** `virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, param_t &pa, func_t &derivs)` [`inline`, `virtual`]

Perform an integration step.

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must generally be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

Reimplemented from [odestep](#).

Definition at line 124 of file `gsl_rkck.h`.

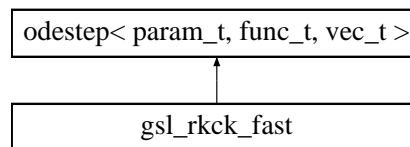
The documentation for this class was generated from the following file:

- `gsl_rkck.h`

## 3.150 gsl\_rkck\_fast Class Template Reference

```
#include <gsl_rkck.h>
```

Inheritance diagram for `gsl_rkck_fast`:



### 3.150.1 Detailed Description

`template<size_t N, class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_rkck_fast< N, param_t, func_t, vec_t, alloc_vec_t, alloc_t >`

Cash-Karp embedded Runge-Kutta formula (GSL) without memory allocation and error checking.

This a fast version of [gsl\\_rkck](#), which is a stepper for a fixed number of ODEs. It ignores the error values returned by the `derivs` argument. The argument `n` to [step\(\)](#) must always be equal to the template parameter `N`, and the vector parameters to `step` must have space allocated for at least `N` elements. No error checking is performed to ensure that this is the case.

Definition at line 211 of file `gsl_rkck.h`.

#### Public Member Functions

- [gsl\\_rkck\\_fast](#) ()
- `virtual ~gsl_rkck_fast` ()
- `virtual int step` (double x, double h, size\_t n, vec\_t &y, vec\_t &dydx, vec\_t &yout, vec\_t &yerr, vec\_t &dydx\_out, param\_t &pa, func\_t &derivs)  
*Perform an integration step.*

#### Protected Attributes

- `alloc_t ao`

*Memory allocator for objects of type `alloc_vec_t`.*

#### Storage for the intermediate steps

- `alloc_vec_t k2`
- `alloc_vec_t k3`
- `alloc_vec_t k4`
- `alloc_vec_t k5`
- `alloc_vec_t k6`
- `alloc_vec_t ytmp`

#### Storage for the coefficients

- `double ah [5]`
- `double b3 [2]`
- `double b4 [3]`
- `double b5 [4]`
- `double b6 [5]`
- `double ec [7]`
- `double b21`
- `double c1`
- `double c3`
- `double c4`
- `double c6`

### 3.150.2 Member Function Documentation

**3.150.2.1** `virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, param_t &pa, func_t &derivs)` [`inline`, `virtual`]

Perform an integration step.

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must generally be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

#### Note:

The value of the parameter `n` must be equal to the template parameter `N`.

Reimplemented from [odestep](#).

Definition at line 303 of file `gsl_rkck.h`.

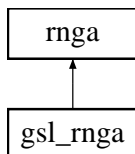
The documentation for this class was generated from the following file:

- `gsl_rkck.h`

## 3.151 gsl\_rnga Class Reference

```
#include <gsl_rnga.h>
```

Inheritance diagram for `gsl_rnga`:



### 3.151.1 Detailed Description

Random number generator (GSL).

If `seed` is zero, or is not given, then the default seed specific to the particular random number generator is used. No virtual functions are used in this class or its parent, [rnga](#). This should be as fast as the original GSL version.

Definition at line 43 of file `gsl_rnga.h`.

#### Public Member Functions

- [gsl\\_rnga](#) (const `gsl_rng_type` \*`gtype`=`gsl_rng_mt19937`)  
*Initialize the random number generator with type `gtype` and the default seed.*
- [gsl\\_rnga](#) (unsigned long int `seed`, const `gsl_rng_type` \*`gtype`=`gsl_rng_mt19937`)  
*Initialize the random number generator with `seed`.*
- const `gsl_rng_type` \* [get\\_type](#) ()  
*Return rng type.*
- double [random](#) ()  
*Return a random number in (0, 1].*
- unsigned long int [get\\_max](#) ()  
*Return the maximum integer for [random\\_int\(\)](#).*
- unsigned long int [random\\_int](#) (unsigned long int `n`=0)  
*Return random integer in [0, max - 1].*
- `gsl_rng` \* [get\\_gsl\\_rng](#) ()  
*Return a pointer to the `gsl_rng` object (deprecated).*

#### Protected Attributes

- `gsl_rng` \* [gr](#)  
*The GSL random number generator.*
- const `gsl_rng_type` \* [rng](#)  
*The GSL random number generator type.*

### 3.151.2 Member Function Documentation

#### 3.151.2.1 `gsl_rng*` [get\\_gsl\\_rng](#) ()

Return a pointer to the `gsl_rng` object (deprecated).

Used in [gsl\\_miser](#) and [gsl\\_anneal](#).

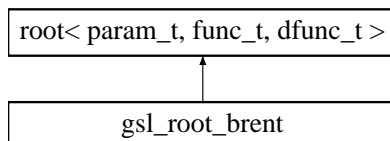
The documentation for this class was generated from the following file:

- `gsl_rnga.h`

## 3.152 gsl\_root\_brent Class Template Reference

```
#include <gsl_root_brent.h>
```

Inheritance diagram for `gsl_root_brent`:





### 3.152.1 Detailed Description

```
template<class param_t, class func_t = funct<void *>, class dfunc_t = func_t> class gsl_root_brent< param_t, func_t,
dfunc_t >
```

One-dimensional root-finding (GSL).

This class finds the [root](#) of a user-specified function. If [test\\_form](#) is 0, then [solve\\_bkt\(\)](#) stops when the size of the bracket is smaller than [root::tolx](#). If [test\\_form](#) is 1, then the function stops when the residual is less than [root::tolf](#). If [test\\_form](#) is 2, then both tests are applied.

#### Todo

There is some duplication in the variables `x_lower`, `x_upper`, `a`, and `b`, which could be removed.

Definition at line 54 of file `gsl_root_brent.h`.

### Public Member Functions

- [gsl\\_root\\_brent](#) ()
- virtual const char \* [type](#) ()  
*Return the type, "gsl\_root\_brent".*
- int [iterate](#) (func\_t &f)  
*Perform an iteration.*
- virtual int [solve\\_bkt](#) (double &x1, double x2, param\_t &pa, func\_t &f)  
*Solve func in region  $x_1 < x < x_2$  returning  $x_1$ .*
- double [get\\_root](#) ()  
*Get the most recent value of the [root](#).*
- double [get\\_lower](#) ()  
*Get the lower limit.*
- double [get\\_upper](#) ()  
*Get the upper limit.*
- int [set](#) (func\_t &ff, double lower, double upper, param\_t &pa)  
*Set the information for the solver.*

### Data Fields

- int [test\\_form](#)  
*The type of convergence test applied: 0, 1, or 2 (default 0).*

### Protected Attributes

- double [root](#)  
*The present solution estimate.*
- double [x\\_lower](#)  
*The present lower limit.*
- double [x\\_upper](#)  
*The present upper limit.*
- param\_t \* [params](#)  
*The function parameters.*

### Storage for solver state

- double [a](#)
- double [b](#)
- double [c](#)

- double [d](#)
- double [e](#)
- double [fa](#)
- double [fb](#)
- double [fc](#)

### 3.152.2 Member Function Documentation

#### 3.152.2.1 int iterate (func\_t &f) [inline]

Perform an iteration.

This function always returns [gsl\\_success](#).

Definition at line 74 of file `gsl_root_brent.h`.

#### 3.152.2.2 virtual int solve\_bkt (double &x1, double x2, param\_t &pa, func\_t &f) [inline, virtual]

Solve `func` in region  $x_1 < x < x_2$  returning  $x_1$ .

Test the bracket size

Test the residual

Test the bracket size and the residual

Reimplemented from [root](#).

Definition at line 186 of file `gsl_root_brent.h`.

#### 3.152.2.3 int set (func\_t &ff, double lower, double upper, param\_t &pa) [inline]

Set the information for the solver.

This function always returns [gsl\\_success](#).

Definition at line 298 of file `gsl_root_brent.h`.

The documentation for this class was generated from the following file:

- `gsl_root_brent.h`

## 3.153 gsl\_root\_stef Class Template Reference

```
#include <gsl_root_stef.h>
```

### 3.153.1 Detailed Description

```
template<class param_t, class func_t, class dfunc_t> class gsl_root_stef< param_t, func_t, dfunc_t >
```

Steffenson equation solver (GSL).

This class finds a [root](#) of a function a derivative. If the derivative is not analytically specified, it is most likely preferable to use of the alternatives, [gsl\\_root\\_brent](#), [cern\\_root](#), or [cern\\_mroot\\_root](#). The function [solve\\_de\(\)](#) performs the solution automatically, and a lower-level GSL-like interface with [set\(\)](#) and [iterate\(\)](#) is also provided.

By default, this solver compares the present value of the [root](#) (*root*) to the previous value (*x*), and returns success if  $|root - x| < tol$ , where  $tol = tol_x + tol_f 2root$ .

If [test\\_residual](#) is set to true, then the solver additionally requires that the absolute value of the function is less than [root::tolf](#).

The original variable `x_2` has been removed as it was unused in the original GSL code.

Definition at line 57 of file gsl\_root\_stef.h.

## Public Member Functions

- [gsl\\_root\\_stef](#) ()
- virtual const char \* [type](#) ()  
*Return the type, "gsl\_root\_stef".*
- int [iterate](#) ()  
*Perform an iteration.*
- virtual int [solve\\_de](#) (double &xx, param\_t &pa, func\_t &fun, dfunc\_t &dfun)  
*Solve func using x as an initial guess using derivatives df.*
- int [set](#) (func\_t &fun, dfunc\_t &dfun, double guess, param\_t &pa)  
*Set the information for the solver.*

## Data Fields

- double [root](#)  
*The present solution estimate.*
- double [tolf2](#)  
*The relative tolerance for subsequent solutions (default  $10^{-12}$ ).*
- bool [test\\_residual](#)  
*True if we should test the residual also (default false).*

## Protected Attributes

- double [f](#)  
*Desc.*
- double [df](#)  
*Desc.*
- double [x\\_1](#)  
*Desc.*
- double [x](#)  
*Desc.*
- int [count](#)  
*Desc.*
- func\_t \* [fp](#)  
*The function to solve.*
- dfunc\_t \* [dfp](#)  
*The derivative.*
- param\_t \* [params](#)  
*The function parameters.*

### 3.153.2 Member Function Documentation

#### 3.153.2.1 int iterate () [inline]

Perform an iteration.

After a successful iteration, [root](#) contains the most recent value of the [root](#).

Definition at line 111 of file gsl\_root\_stef.h.

### 3.153.2.2 int set (func\_t &fun, dfunc\_t &dfun, double guess, param\_t &pa) [inline]

Set the information for the solver.

Set the function, the derivative, the initial guess and the parameters.

Definition at line 222 of file gsl\_root\_stef.h.

The documentation for this class was generated from the following file:

- gsl\_root\_stef.h

## 3.154 gsl\_series Class Reference

```
#include <gsl_series.h>
```

### 3.154.1 Detailed Description

Series acceleration by Levin u-transform (GSL).

Given an array of terms in a sum, this attempts to evaluate the entire sum with an estimate of the error.

#### Todo

Covert to use a more general vector

Definition at line 43 of file gsl\_series.h.

### Public Member Functions

- [gsl\\_series](#) (int size=1)  
*size is the number of terms in the series*
- double [series\\_accel](#) (double \*x, double &err)  
*Return the accelerated sum of the series with a simple error estimate.*
- double [series\\_accel\\_err](#) (double \*x, double &err)  
*Return the accelerated sum of the series with an accurate error estimate.*
- int [set\\_size](#) (int new\_size)  
*Set the number of terms.*

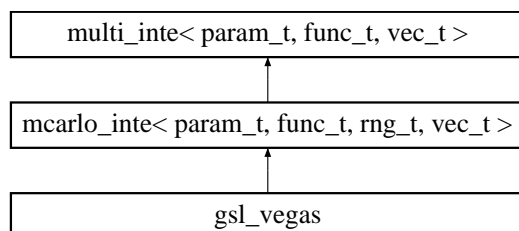
The documentation for this class was generated from the following file:

- gsl\_series.h

## 3.155 gsl\_vegas Class Template Reference

```
#include <gsl_vegas.h>
```

Inheritance diagram for gsl\_vegas::



### 3.155.1 Detailed Description

`template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_view> class gsl_vegas< param_t, func_t, rng_t, vec_t >`

Multidimensional integration using plain Vegas Carlo (GSL).

Definition at line 42 of file `gsl_vegas.h`.

#### Public Member Functions

- virtual `~gsl_vegas()`
- virtual `int minteg_err(func_t &func, size_t ndim, const vec_t &a, const vec_t &b, param_t &pa, double &res, double &err)`  
*Integrate function `func` from  $x=a$  to  $x=b$ .*
- virtual `const char * type()`  
*Return string denoting type ("`gsl_vegas`").*

#### Static Protected Member Functions

- static `double gsl_func(double *x, size_t dim, void *pa)`  
*The `GSL` function pointer.*

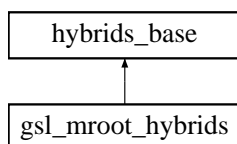
The documentation for this class was generated from the following file:

- `gsl_vegas.h`

## 3.156 hybrids\_base Class Reference

`#include <gsl_mroot_hybrids_b.h>`

Inheritance diagram for `hybrids_base`:



### 3.156.1 Detailed Description

Base functions for `gsl_mroot_hybrids`.

This is a trivial recasting of the functions that were in file scope in the `GSL` version of the hybrids solver.

#### Todo

Document the individual functions for this class

Definition at line 46 of file `gsl_mroot_hybrids_b.h`.

## Protected Member Functions

- double [enorm](#) (const gsl\_vector \*f)  
*Compute the norm of  $f$ .*
- double [scaled\\_enorm](#) (const gsl\_vector \*d, const gsl\_vector \*f)  
*Compute the norm of  $\vec{f} \cdot \vec{d}$ .*
- double [enorm\\_sum](#) (const gsl\_vector \*a, const gsl\_vector \*b)  
*Compute the norm of  $\vec{a} + \vec{b}$ .*
- void [compute\\_wv](#) (const gsl\_vector \*qtdf, const gsl\_vector \*rdx, const gsl\_vector \*dx, const gsl\_vector \*diag, double pnorm, gsl\_vector \*w, gsl\_vector \*v)  
*Desc.*
- void [compute\\_df](#) (const gsl\_vector \*f\_trial, const gsl\_vector \*f, gsl\_vector \*df)  
*Desc.*
- void [compute\\_diag](#) (const gsl\_matrix \*J, gsl\_vector \*diag)  
*Desc.*
- void [update\\_diag](#) (const gsl\_matrix \*J, gsl\_vector \*diag)  
*Desc.*
- double [compute\\_delta](#) (gsl\_vector \*diag, gsl\_vector \*x)  
*Desc.*
- double [compute\\_actual\\_reduction](#) (double fnorm, double fnorm1)  
*Desc.*
- double [compute\\_predicted\\_reduction](#) (double fnorm, double fnorm1)  
*Desc.*
- void [compute\\_qtf](#) (const gsl\_matrix \*q, const gsl\_vector \*f, gsl\_vector \*qtf)  
*Compute  $Q^T f$ .*
- void [compute\\_rdx](#) (const gsl\_matrix \*r, const gsl\_vector \*dx, gsl\_vector \*rdx)  
*Desc.*
- void [compute\\_trial\\_step](#) (gsl\_vector \*x, gsl\_vector \*dx, gsl\_vector \*x\_trial)  
*Desc.*
- int [newton\\_direction](#) (const gsl\_matrix \*r, const gsl\_vector \*qtf, gsl\_vector \*p)  
*Desc.*
- void [gradient\\_direction](#) (const gsl\_matrix \*r, const gsl\_vector \*qtf, const gsl\_vector \*diag, gsl\_vector \*g)  
*Desc.*
- void [minimum\\_step](#) (double gnorm, const gsl\_vector \*diag, gsl\_vector \*g)  
*Desc.*
- void [compute\\_Rg](#) (const gsl\_matrix \*r, const gsl\_vector \*gradient, gsl\_vector \*Rg)  
*Desc.*
- void [scaled\\_addition](#) (double alpha, gsl\_vector \*newton, double beta, gsl\_vector \*gradient, gsl\_vector \*p)  
*Desc.*
- int [dogleg](#) (const gsl\_matrix \*r, const gsl\_vector \*qtf, const gsl\_vector \*diag, double delta, gsl\_vector \*newton, gsl\_vector \*gradient, gsl\_vector \*p)  
*Take a dogleg step.*

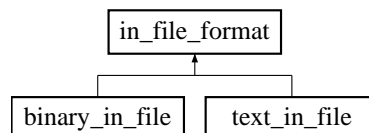
The documentation for this class was generated from the following file:

- `gsl_mroot_hybrids_b.h`

## 3.157 in\_file\_format Class Reference

```
#include <file_format.h>
```

Inheritance diagram for in\_file\_format::



### 3.157.1 Detailed Description

Abstract base class for input file formats.

Definition at line 101 of file file\_format.h.

#### Public Member Functions

- virtual `~in_file_format()`
- virtual int `bool_in` (bool &dat, std::string name="")=0  
*Input a bool variable.*
- virtual int `char_in` (char &dat, std::string name="")=0  
*Input a char variable.*
- virtual int `double_in` (double &dat, std::string name="")=0  
*Input a double variable.*
- virtual int `float_in` (float &dat, std::string name="")=0  
*Input a float variable.*
- virtual int `int_in` (int &dat, std::string name="")=0  
*Input an int variable.*
- virtual int `long_in` (unsigned long int &dat, std::string name="")=0  
*Input an long variable.*
- virtual int `string_in` (std::string &dat, std::string name="")=0  
*Input a string variable.*
- virtual int `word_in` (std::string &dat, std::string name="")=0  
*Input a word variable.*
- virtual int `start_object` (std::string &type, std::string &name)=0  
*Start object input.*
- virtual int `skip_object` ()=0  
*Skip the present object for the next call to read\_type().*
- virtual int `end_object` ()=0  
*End object input.*
- virtual int `init_file` ()=0  
*Read initialization.*
- virtual int `clean_up` ()=0  
*Finish file input.*

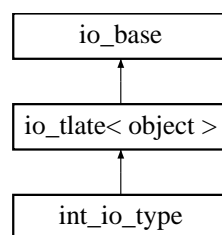
The documentation for this class was generated from the following file:

- file\_format.h

## 3.158 int\_io\_type Class Reference

```
#include <collection.h>
```

Inheritance diagram for int\_io\_type::



**3.158.1 Detailed Description**

I/O object for int variables.

Definition at line 1742 of file collection.h.

**Public Member Functions**

- [int\\_io\\_type](#) (const char \*)  
*Desc.*
- [int\\_io\\_type](#) ()
- int [addi](#) ([collection](#) &co, std::string name, int x, bool overwrt=true)  
*Add a int to a [collection](#).*
- int [geti](#) ([collection](#) &co, std::string tname)  
*Get a int from a [collection](#).*
- int [get\\_def](#) ([collection](#) &co, std::string tname, int &op, int def=0)  
*Get a int from a [collection](#).*

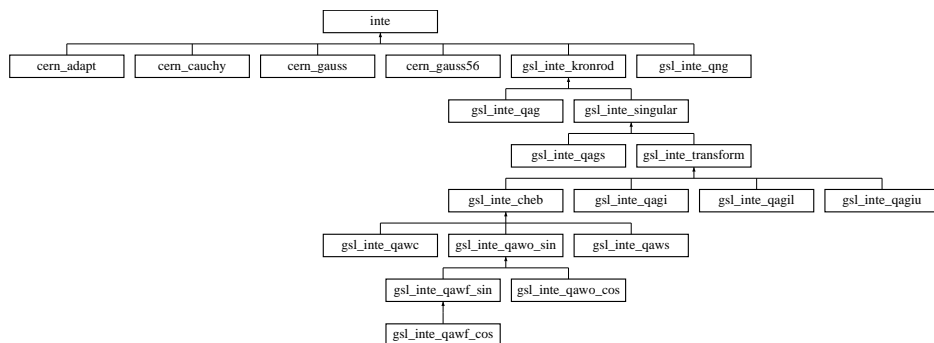
The documentation for this class was generated from the following file:

- collection.h

**3.159    inte Class Template Reference**

```
#include <inte.h>
```

Inheritance diagram for inte::

**3.159.1 Detailed Description**

```
template<class param_t, class func_t> class inte< param_t, func_t >
```

Base integration class.

Definition at line 35 of file inte.h.

**Public Member Functions**

- [inte](#) ()
- virtual [~inte](#) ()
- virtual double [integ](#) (func\_t &func, double a, double b, param\_t &pa)  
*Integrate function func from a to b.*



- virtual int [integ\\_err](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err)  
*Integrate function func from a to b and place the result in res and the error in err.*
- double [get\\_error](#) ()  
*Return the error in the result from the last call to [integ\(\)](#).*
- virtual const char \* [type](#) ()  
*Return string denoting type ("inte").*

### Data Fields

- int [verbose](#)  
*Verbosity.*
- double [tolf](#)  
*The maximum relative uncertainty in the value of the integral (default  $10^{-8}$ ).*
- double [tolx](#)  
*The maximum absolute uncertainty in the value of the integral (default  $10^{-8}$ ).*

### Protected Attributes

- double [intererror](#)  
*The uncertainty for the last integration computation.*

## 3.159.2 Member Function Documentation

**3.159.2.1** virtual int [integ\\_err](#) (func\_t &func, double a, double b, param\_t &pa, double &res, double &err) [inline, virtual]

Integrate function func from a to b and place the result in res and the error in err.

Ideally, if this function succeeds, then err should be less than or close to [tolf](#).

Reimplemented in [cern\\_adapt](#), [cern\\_cauchy](#), [cern\\_gauss](#), [cern\\_gauss56](#), [gsl\\_inte\\_qag](#), [gsl\\_inte\\_qagi](#), [gsl\\_inte\\_qagil](#), [gsl\\_inte\\_qagiu](#), [gsl\\_inte\\_qags](#), [gsl\\_inte\\_qawc](#), [gsl\\_inte\\_qawf\\_sin](#), [gsl\\_inte\\_qawf\\_cos](#), [gsl\\_inte\\_qawo\\_sin](#), [gsl\\_inte\\_qawo\\_cos](#), [gsl\\_inte\\_qaws](#), and [gsl\\_inte\\_qng](#).

Definition at line 74 of file inte.h.

**3.159.2.2** double [get\\_error](#) () [inline]

Return the error in the result from the last call to [integ\(\)](#).

This will quietly return zero if no integrations have been performed.

Definition at line 85 of file inte.h.

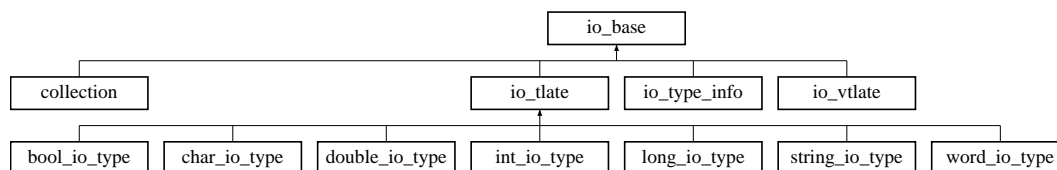
The documentation for this class was generated from the following file:

- inte.h

## 3.160 io\_base Class Reference

```
#include <collection.h>
```

Inheritance diagram for io\_base::



### 3.160.1 Detailed Description

I/O base class.

This class is necessary so that the `collection` method source code and the `io_base` method source code doesn't have to go in header files.

#### Todo

Should the `remove()` functions be moved to class `collection`?

Definition at line 98 of file `collection.h`.

### Public Member Functions

- `io_base` (int sw=0)  
*Create a new I/O object.*
- `io_base` (const char \*t)  
*Create a new object only if an I/O object for type t is not yet present.*

### Functions to be overloaded in descendants of io\_base

- virtual const char \* `type` ()  
*Return the type of an object.*
- virtual bool `has_static_data` ()  
*If true, then the object contains static data.*

### Functions useful for in in() and out()

- virtual int `pointer_in` (cinput \*co, in\_file\_format \*ins, void \*\*pp, std::string &stype)  
*Input a pointer.*
- virtual int `pointer_out` (coutput \*co, out\_file\_format \*outs, void \*ptr, std::string stype)  
*Output an object to outs of type stype.*

### Protected Member Functions

- virtual int `stat_in_noobj` (cinput \*co, in\_file\_format \*ins)  
*Automatically create an object for stat\_in.*
- virtual int `stat_out_noobj` (coutput \*co, out\_file\_format \*outs)  
*Automatically create an object for stat\_out.*
- virtual int `in_wrapper` (cinput \*co, in\_file\_format \*ins, void \*&vp)  
*Allocate memory and input an object.*
- virtual int `in_wrapper` (cinput \*co, in\_file\_format \*ins, void \*&vp, int &sz)  
*Allocate memory and input an array of objects.*
- virtual int `in_wrapper` (cinput \*co, in\_file\_format \*ins, void \*&vp, int &sz, int &sz2)  
*Allocate memory and input a 2-d array of objects.*
- virtual int `out_wrapper` (coutput \*co, out\_file\_format \*outs, void \*vp, int sz, int sz2)  
*Internal function to output an object (or an array or 2-d array).*
- virtual int `object_in_void` (cinput \*cin, in\_file\_format \*ins, void \*op, std::string &name)

*Input an object (no memory allocation).*

- virtual int `object_in_void` (`cinput *cin`, `in_file_format *ins`, void \*op, int sz, std::string &name)  
*Input an array of objects (no memory allocation).*
- virtual int `object_in_void` (`cinput *cin`, `in_file_format *ins`, void \*op, int sz, int sz2, std::string &name)  
*Input a 2-d array of objects (no memory allocation).*
- virtual int `object_in_mem_void` (`cinput *cin`, `in_file_format *ins`, void \*&op, std::string &name)  
*Input an object (no memory allocation).*
- virtual int `object_in_mem_void` (`cinput *cin`, `in_file_format *ins`, void \*&op, int &sz, std::string &name)  
*Input an array of objects (no memory allocation).*
- virtual int `object_in_mem_void` (`cinput *cin`, `in_file_format *ins`, void \*&op, int &sz, int &sz2, std::string &name)  
*Input a 2-d array of objects (no memory allocation).*
- virtual int `object_out_void` (`coutput *cout`, `out_file_format *outs`, void \*op, int sz, int sz2, std::string name="")  
*Output an object, an array of objects, or a 2-d array of objects.*

### Functions to remove the memory that was allocated for an object

- virtual int `remove` (void \*vp)  
*Remove the memory for an object.*
- virtual int `remove_arr` (void \*vp)  
*Remove the memory for an array of objects.*
- virtual int `remove_2darr` (void \*vp, int sz)  
*Remove the memory for a 2-dimensional array of objects.*

### Protected Attributes

- int `sw_store`  
*Store the value of sw given in the constructor so that we know if we need to remove the type in the destructor.*

### Static Protected Attributes

- static class `io_manager * iom`  
*A pointer to the type manager.*
- static int `objs_count`  
*A count of the number of objects.*

## 3.160.2 Constructor & Destructor Documentation

### 3.160.2.1 io\_base (int sw = 0)

Create a new I/O object.

If `sw` is different from zero, then the type will not be added to the `io_manager`. This is useful if you want an object to be its own I/O class, in which case you may want to make sure that the `io_manager` only tries to add the type once. There is no need to have an I/O object for every instance of a particular type.

## 3.160.3 Member Function Documentation

### 3.160.3.1 virtual int pointer\_out (coutput \*co, out\_file\_format \*outs, void \*ptr, std::string stype) [virtual]

Output an object to `outs` of type `stype`.

This is useful for to output a pointer to an object in the `out()` or `stat_out()` functions for a class. The data for the object which is pointed to is separate from the object and is only referred to once if more than one objects point to it.

The documentation for this class was generated from the following file:

- `collection.h`

## 3.161 io\_manager Class Reference

```
#include <collection.h>
```

### 3.161.1 Detailed Description

Manage I/O type information.

This class is automatically created, utilized, and destroyed by [io\\_base](#).

Definition at line 257 of file [collection.h](#).

#### Public Member Functions

- [int add\\_type](#) ([io\\_base](#) \*iop)  
*Add a type to the list.*
- [int is\\_type](#) ([io\\_base](#) \*iop)  
*Return 0 if iop points to a valid type.*
- [int add\\_type](#) ([io\\_base](#) \*iop, const char \*t)  
*Add type iop to the manager assuming the type name t.*
- [int remove\\_type](#) ([io\\_base](#) \*iop)  
*Remove a type from the list.*

#### Protected Types

- `typedef std::vector< io\_base * >::iterator titer`  
*A useful definition for iterating through types.*

#### Protected Member Functions

- [io\\_base](#) \* [get\\_ptr](#) (std::string stype)  
*Get a pointer to type stype.*
- [io\\_manager](#) ()  
*Empty constructor.*

#### Protected Attributes

- `std::vector< io\_base * > tlist`  
*The list of types in the form of [io\\_base](#) pointers.*

### 3.161.2 Member Function Documentation

#### 3.161.2.1 int add\_type (io\_base \* iop)

Add a type to the list.

Unfortunately, [add\\_type\(\)](#) cannot ensure that no type is added more than once, since the type is not specified until the entire constructor hierarchy has been executed and [add\\_type\(\)](#) is called at the top of this hierarchy.

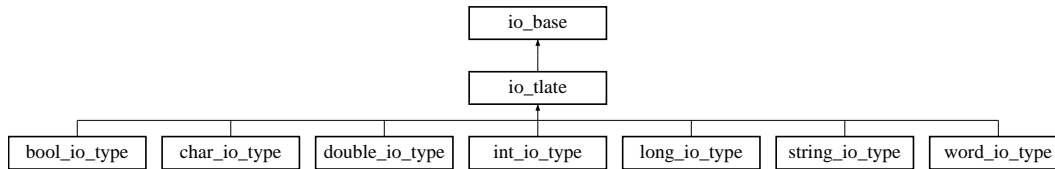
The documentation for this class was generated from the following file:

- [collection.h](#)
-

## 3.162 io\_tlate Class Template Reference

```
#include <collection.h>
```

Inheritance diagram for io\_tlate::



### 3.162.1 Detailed Description

**template<class object> class io\_tlate< object >**

A template for adding I/O classes (documents template [io\\_tlate](#)).

Note that the generic interface here only works with pointers, not with the actual objects themselves. This is important, because it avoids the problem of I/O for an object with private copy and assignment operators. For basic types (bool, char, double, int, etc.), some additional [add\(\)](#) and [get\(\)](#) functions are defined.

Definition at line 1059 of file collection.h.

### Public Member Functions

- [io\\_tlate](#) ()  
*Create an I/O class for type object.*
- [io\\_tlate](#) (const char \*t)  
*Create an I/O class for type object only if another object of type t is not yet present.*
- template<>  
int **input** (cinput \*co, in\_file\_format \*ins, bool \*dp)
- template<>  
int **output** (coutput \*co, out\_file\_format \*outs, bool \*dp)
- template<>  
const char \* [type](#) ()  
*Return the type of an object.*
- template<>  
int **input** (cinput \*co, in\_file\_format \*ins, char \*dp)
- template<>  
int **output** (coutput \*co, out\_file\_format \*outs, char \*dp)
- template<>  
const char \* [type](#) ()  
*Return the type of an object.*
- template<>  
int **input** (cinput \*co, in\_file\_format \*ins, double \*dp)
- template<>  
int **output** (coutput \*co, out\_file\_format \*outs, double \*dp)
- template<>  
const char \* [type](#) ()  
*Return the type of an object.*
- template<>  
int **input** (cinput \*co, in\_file\_format \*ins, int \*dp)
- template<>  
int **output** (coutput \*co, out\_file\_format \*outs, int \*dp)

- `template<>`  
`const char * type ()`  
*Return the type of an object.*
- `template<>`  
`int input (cinput *co, in_file_format *ins, unsigned long int *dp)`
- `template<>`  
`int output (coutput *co, out_file_format *outs, unsigned long int *dp)`
- `template<>`  
`const char * type ()`  
*Return the type of an object.*
- `template<>`  
`int input (cinput *co, in_file_format *ins, std::string *dp)`
- `template<>`  
`int output (coutput *co, out_file_format *outs, std::string *dp)`
- `template<>`  
`const char * type ()`  
*Return the type of an object.*
- `template<>`  
`int input (cinput *co, o2scl::in_file_format *ins, table *ta)`
- `template<>`  
`int output (coutput *co, o2scl::out_file_format *outs, table *at)`
- `template<>`  
`const char * type ()`  
*Return the type of an object.*
- `template<>`  
`int input (cinput *co, in_file_format *ins, gsl_series *gs)`
- `template<>`  
`int output (coutput *co, out_file_format *outs, gsl_series *gs)`
- `template<>`  
`const char * type ()`  
*Return the type of an object.*

### Functions to be overloaded

These functions should be overloaded in all descendants of `io_tlate`.

- `virtual const char * type ()`  
*The name of the type to be processed.*
- `virtual int input (cinput *cin, in_file_format *ins, object *op)`  
*Method for reading an object from ins.*
- `virtual int output (coutput *cout, out_file_format *outs, object *op)`  
*Method for writing an object to outs.*

### Functions to be overloaded for static data

These functions should be overloaded in all descendants of `io_tlate` which control I/O for classes which contain static data.

- `virtual bool has_static_data ()`  
*true if the object contains static I/O data*
- `virtual int stat_input (cinput *cin, in_file_format *ins, object *op)`  
*Method for reading static data for an object from ins.*
- `virtual int stat_output (coutput *cout, out_file_format *outs, object *op)`  
*Method for writing static data for an object to outs.*

### Input functions

- `virtual int object_in (cinput *cin, in_file_format *ins, object *op, std::string &name)`  
*Read an object from ins.*
- `virtual int object_in (cinput *cin, in_file_format *ins, object *op, int sz, std::string &name)`  
*Read an array of objects from ins.*

- virtual int [object\\_in](#) (cinput \*cin, in\_file\_format \*ins, object \*\*op, int sz, int sz2, std::string &name)  
*Read a 2-d array of objects from ins.*
- template<size\_t N>  
int [object\\_in](#) (cinput \*co, in\_file\_format \*ins, object op[ ][N], int sz, std::string &name)  
*Create memory for a 2-d array of objects and read it from ins.*
- virtual int [object\\_in\\_mem](#) (cinput \*cin, in\_file\_format \*ins, object \*&op, std::string &name)  
*Create memory for an object and read it from ins.*
- virtual int [object\\_in\\_mem](#) (cinput \*cin, in\_file\_format \*ins, object \*&op, int &sz, std::string &name)  
*Create memory for an object and read it from ins.*
- virtual int [object\\_in\\_mem](#) (cinput \*cin, in\_file\_format \*ins, object \*\*&op, int &sz, int &sz2, std::string &name)  
*Create memory for an object and read it from ins.*
- template<size\_t N>  
int [object\\_in\\_mem](#) (cinput \*co, in\_file\_format \*ins, object op[ ][N], int &sz, std::string &name)  
*Create memory for a 2-d array of objects and read it from ins.*

### Output functions

- virtual int [object\\_out](#) (coutput \*cout, out\_file\_format \*outs, object \*op, int sz=0, std::string name="")  
*Output an object (or an array of objects) to outs.*
- virtual int [object\\_out](#) (coutput \*cout, out\_file\_format \*outs, object \*\*op, int sz, int sz2, std::string name="")  
*Output an object (or an array of objects) to outs.*
- template<size\_t N>  
int [object\\_out](#) (coutput \*cout, out\_file\_format \*outs, object op[ ][N], int sz, std::string name="")  
*Output a 2-d array of objects to outs.*

### Memory allocation

- virtual int [mem\\_alloc](#) (object \*&op)  
*Create memory for an object.*
- virtual int [mem\\_alloc\\_arr](#) (object \*&op, int sz)  
*Create memory for an object.*
- virtual int [mem\\_alloc\\_2darr](#) (object \*\*&op, int sz, int sz2)  
*Create memory for an object.*

### Add and get objects from a collection

- int [add](#) (collection &coll, std::string name, object \*op, int sz=0, bool overwrt=true, bool owner=false)  
*Add an object(s) to a [collection](#).*
- int [add\\_2darray](#) (collection &coll, std::string name, object \*\*op, int sz, int sz2, bool overwrt=true, bool owner=false)  
*Add an object(s) to a [collection](#).*
- int [get](#) (collection &coll, std::string tname, object \*&op)  
*Get an object(s) from a [collection](#).*
- int [get](#) (collection &co, std::string tname, object \*&op, int &sz)  
*Get an object(s) from a [collection](#).*
- int [get](#) (collection &co, std::string tname, object \*\*&op, int &sz, int &sz2)  
*Get an object(s) from a [collection](#).*

### Other functions

- virtual int [mem\\_free](#) (object \*op)  
*Free the memory associated with an object.*
- virtual int [mem\\_free\\_arr](#) (object \*op)  
*Free the memory associated with an array of objects.*
- virtual int [mem\\_free\\_2darr](#) (object \*\*op, int sz)  
*Free the memory associated with a 2-d array of objects.*

## Protected Member Functions

- virtual int [object\\_in\\_void](#) (cinput \*cin, in\_file\_format \*ins, void \*op, std::string &name)  
*Input an object (no memory allocation).*
- virtual int [object\\_in\\_void](#) (cinput \*cin, in\_file\_format \*ins, void \*op, int sz, std::string &name)  
*Input an array of objects (no memory allocation).*
- virtual int [object\\_in\\_void](#) (cinput \*cin, in\_file\_format \*ins, void \*op, int sz, int sz2, std::string &name)  
*Input a 2-d array of objects (no memory allocation).*
- virtual int [object\\_in\\_mem\\_void](#) (cinput \*cin, in\_file\_format \*ins, void \*&vp, std::string &name)  
*Input an object (no memory allocation).*
- virtual int [object\\_in\\_mem\\_void](#) (cinput \*cin, in\_file\_format \*ins, void \*&vp, int &sz, std::string &name)  
*Input an array of objects (no memory allocation).*
- virtual int [object\\_in\\_mem\\_void](#) (cinput \*cin, in\_file\_format \*ins, void \*&vp, int &sz, int &sz2, std::string &name)  
*Input a 2-d array of objects (no memory allocation).*
- virtual int [object\\_out\\_void](#) (coutput \*cout, out\_file\_format \*outs, void \*op, int sz=0, int sz2=0, std::string name="")  
*Output an object, an array of objects, or a 2-d array of objects.*
- virtual int [stat\\_in\\_noobj](#) (cinput \*cin, in\_file\_format \*ins)  
*Automatically create an object for stat\_in.*
- virtual int [stat\\_out\\_noobj](#) (coutput \*cout, out\_file\_format \*outs)  
*Automatically create an object for stat\_out.*
- int [in\\_wrapper](#) (cinput \*cin, in\_file\_format \*ins, void \*&vp)  
*Allocate memory and input an object.*
- int [in\\_wrapper](#) (cinput \*cin, in\_file\_format \*ins, void \*&vp, int &sz)  
*Allocate memory and input an array of objects.*
- int [in\\_wrapper](#) (cinput \*cin, in\_file\_format \*ins, void \*&vp, int &sz, int &sz2)  
*Allocate memory and input a 2-d array of objects.*
- int [out\\_wrapper](#) (coutput \*cout, out\_file\_format \*outs, void \*vp, int sz, int sz2)  
*Internal function to output an object (or an array or 2-d array).*
- virtual int [remove](#) (void \*vp)  
*Remove the memory for an object.*
- virtual int [remove\\_arr](#) (void \*vp)  
*Remove the memory for an array of objects.*
- virtual int [remove\\_2darr](#) (void \*vp, int sz)  
*Remove the memory for a 2-dimensional array of objects.*
- virtual int [stat\\_in\\_wrapper](#) (cinput \*cin, in\_file\_format \*ins, void \*vp)  
*Static input for an object.*
- virtual int [stat\\_out\\_wrapper](#) (coutput \*cout, out\_file\_format \*outs, void \*vp)  
*Static output for an object.*

### 3.162.2 Member Function Documentation

#### 3.162.2.1 virtual int stat\_input (cinput \*cin, in\_file\_format \*ins, object \*op) [inline, virtual]

Method for reading static data for an object from ins.

One must be careful about objects which set the static data in their constructors. An object is automatically created in order to read its static data. This means that if the static data is set in the constructor, then possibly useful information will be overwritten through the creation of this temporary object.

If one needs to set static data in the constructor of a singleton object, then the create() and [remove\(\)](#) functions should be empty and a separate pointer to the singleton should be provided instead of void \*vp.

This is only used if [has\\_static\\_data\(\)](#) returns true;

Definition at line 1119 of file collection.h.



### 3.162.2.2 int object\_in\_mem (cinput \*co, in\_file\_format \*ins, object op[ ][N], int &sz, std::string &name) [inline]

Create memory for a 2-d array of objects and read it from ins.

Note that you must specify in advance the size N.

Definition at line 1445 of file collection.h.

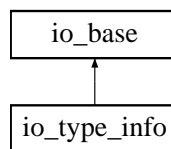
The documentation for this class was generated from the following file:

- collection.h

## 3.163 io\_type\_info Class Reference

```
#include <collection.h>
```

Inheritance diagram for io\_type\_info::



### 3.163.1 Detailed Description

User interface to provide I/O type information.

Definition at line 324 of file collection.h.

#### Public Member Functions

##### Type manipulation

- int [is\\_type](#) (std::string stype)  
*Return 0 if stype is a valid I/O type.*
- int [remove\\_type](#) (std::string stype)  
*Remove stype from the list of valid I/O types.*
- virtual int [clear\\_types](#) ()  
*Remove all types in the list of valid I/O types.*
- void [type\\_summary](#) (std::ostream \*outs, bool pointers=false)  
*Print a summary of valid types to the outs stream.*
- int [add\\_type](#) (io\_base \*iop)  
*Add an I/O type to the list.*

#### Protected Types

- typedef std::vector< [io\\_base](#) \* >::iterator [titer](#)  
*A useful definition for iterating through types.*

#### Protected Member Functions

- int [static\\_fout](#) (coutput \*co, out\_file\_format \*out)  
*Output the static information for the I/O types.*
- int [static\\_fout\\_restricted](#) (coutput \*co, out\_file\_format \*out, std::set< std::string, [string\\_comp](#) > list)  
*Output the static information for the I/O types not in the list.*

### 3.163.2 Member Function Documentation

#### 3.163.2.1 int remove\_type (std::string *stype*)

Remove *stype* from the list of valid I/O types.

This method is dangerous, as it can't check to ensure that no [collection](#) has remaining objects of the type to be removed.

#### 3.163.2.2 virtual int clear\_types () [virtual]

Remove all types in the list of valid I/O types.

This method is dangerous as it doesn't ensure that all collections are empty.

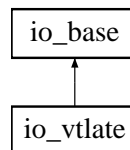
The documentation for this class was generated from the following file:

- collection.h

## 3.164 io\_vtlate Class Template Reference

```
#include <collection.h>
```

Inheritance diagram for io\_vtlate:



### 3.164.1 Detailed Description

```
template<class object> class io_vtlate< object >
```

A template for adding I/O classes.

Definition at line 983 of file collection.h.

#### Public Member Functions

##### Functions to be overloaded

*These functions should be overloaded in all descendants of [io\\_tlate](#).*

- virtual const char \* [type](#) ()  
*The name of the type to be processed.*
- virtual int [input](#) ([cinput](#) \*cin, [in\\_file\\_format](#) \*ins, object \*op)  
*Method for reading an object from ins.*
- virtual int [output](#) ([coutput](#) \*cout, [out\\_file\\_format](#) \*outs, object \*op)  
*Method for writing an object to outs.*

##### Functions to be overloaded for static data

*These functions should be overloaded in all descendants of [io\\_tlate](#) which control I/O for classes which contain static data.*

- virtual bool [has\\_static\\_data](#) ()  
*true if the object contains static I/O data*
- virtual int [stat\\_input](#) ([cinput](#) \*cin, [in\\_file\\_format](#) \*ins, object \*op)

*Method for reading static data for an object from ins.*

- virtual int [stat\\_output](#) (coutput \*cout, out\_file\_format \*outs, object \*op)

*Method for writing static data for an object to outs.*

### 3.164.2 Member Function Documentation

#### 3.164.2.1 virtual int stat\_input (cinput \*cin, in\_file\_format \*ins, object \*op) [inline, virtual]

Method for reading static data for an object from ins.

One must be careful about objects which set the static data in their constructors. An object is automatically created in order to read its static data. This means that if the static data is set in the constructor, then possibly useful information will be overwritten through the creation of this temporary object.

If one needs to set static data in the constructor of a singleton object, then the create() and remove() functions should be empty and a separate pointer to the singleton should be provided instead of void \*vp.

This is only used if [has\\_static\\_data\(\)](#) returns true;

Definition at line 1034 of file collection.h.

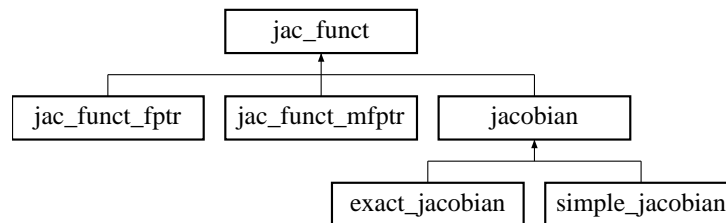
The documentation for this class was generated from the following file:

- collection.h

## 3.165 jac\_func Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for jac\_func::



### 3.165.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view, class mat_t = omatrix_view> class jac_func< param_t, vec_t, mat_t >
```

Base for a square Jacobian where J is computed at x given y=f(x).

Compute

$$J(i, j) = \frac{\partial f_i}{\partial x_j}$$

The vec\_t objects in operator() could have been written to be const, but they are not const so that they can be used as temporary workspace. They are restored to their original values before operator() exits.

Definition at line 51 of file jacobian.h.

### Public Member Functions

- [jac\\_func](#) ()

- virtual `~jac_funct()`
- virtual int `operator()` (size\_t nv, vec\_t &x, vec\_t &y, mat\_t &j, param\_t &pa)  
*The operator().*

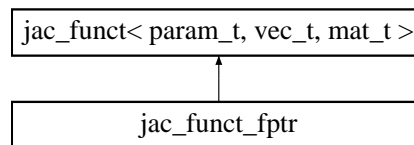
The documentation for this class was generated from the following file:

- jacobian.h

## 3.166 `jac_funct_fptr` Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for `jac_funct_fptr`:



### 3.166.1 Detailed Description

**template<class param\_t, class vec\_t = ovector\_view, class mat\_t = omatrix\_view> class `jac_funct_fptr`< param\_t, vec\_t, mat\_t >**

Function pointer to [jacobian](#).

Definition at line 80 of file jacobian.h.

#### Public Member Functions

- `jac_funct_fptr` (int(\*fp)(size\_t nv, vec\_t &x, vec\_t &y, mat\_t &j, param\_t &pa))  
*Specify the function pointer.*
- virtual int `operator()` (size\_t nv, vec\_t &x, vec\_t &y, mat\_t &j, param\_t &pa)  
*The operator().*

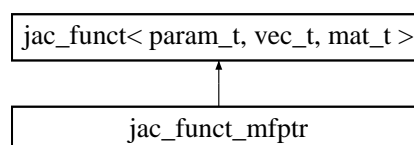
The documentation for this class was generated from the following file:

- jacobian.h

## 3.167 `jac_funct_mfptr` Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for `jac_funct_mfptr`:



### 3.167.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_view, class mat_t = omatrix_view> class jac_funcnt_mfptr< tclass, param_t, vec_t, mat_t >
```

Member function pointer to a Jacobian.

Definition at line 123 of file jacobian.h.

### Public Member Functions

- [jac\\_funcnt\\_mfptr](#) (tclass \*tp, int(tclass::\*fp)(size\_t nv, vec\_t &x, vec\_t &y, mat\_t &j, param\_t &pa))  
*Specify the member function pointer.*
- virtual [~jac\\_funcnt\\_mfptr](#) ()
- virtual int [operator\(\)](#) (size\_t nv, vec\_t &x, vec\_t &y, mat\_t &j, param\_t &pa)  
*The operator().*

### Protected Attributes

- int(tclass::\* [fptr](#)) (size\_t nv, vec\_t &x, vec\_t &y, mat\_t &j, param\_t &pa)  
*Desc.*
- tclass \* [tptr](#)  
*Desc.*

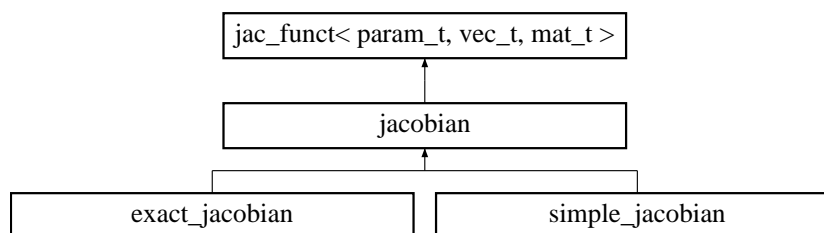
The documentation for this class was generated from the following file:

- jacobian.h

## 3.168 jacobian Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for jacobian::



### 3.168.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view> class jacobian< param_t, func_t, vec_t, mat_t >
```

Base for providing a numerical [jacobian](#).

Definition at line 174 of file jacobian.h.

## Public Member Functions

- `jacobian()`
- `virtual ~jacobian()`
- `virtual int set_function(func_t &f)`  
*Set the function to compute the Jacobian of.*
- `virtual int operator()(size_t nv, vec_t &x, vec_t &y, mat_t &j, param_t &pa)`  
*The operator().*

## Protected Attributes

- `func_t * func`  
*A pointer to the user-specified function.*

## Private Member Functions

- `jacobian` (const `jacobian` &)
- `jacobian` & `operator=` (const `jacobian` &)

The documentation for this class was generated from the following file:

- `jacobian.h`

## 3.169 lanczos Class Template Reference

```
#include <lanczos.h>
```

### 3.169.1 Detailed Description

```
template<class vec_t, class mat_t, class alloc_vec_t, class alloc_t> class lanczos< vec_t, mat_t, alloc_vec_t, alloc_t >
```

Lanczos diagonalization.

This is useful for approximating the largest eigenvalues of a symmetric matrix.

The vector and matrix types can be any type which provides access via `operator[]`, given suitably constructed allocation types.

The tridiagonalization routine was rewritten from the EISPACK routines `imtql1.f` (but uses `gsl_hypot()` instead of `pythag.f`).

### Idea for future

The function `eigen_tdiag()` automatically sorts the eigenvalues, which may not be necessary.

Definition at line 52 of file `lanczos.h`.

## Public Member Functions

- `lanczos()`
- `int eigenvalues` (size\_t size, mat\_t &mat, size\_t n\_iter, vec\_t &eigen, vec\_t &diag, vec\_t &off\_diag)  
*Approximate the largest eigenvalues of a symmetric matrix mat using the Lanczos method.*
- `int eigen_tdiag` (size\_t n, vec\_t &diag, vec\_t &off\_diag)  
*In-place diagonalization of a tri-diagonal matrix.*

## Data Fields

- size\_t [td\\_iter](#)  
*Number of iterations for finding the eigenvalues of the tridiagonal matrix (default 30).*
- size\_t [td\\_lasteval](#)  
*The index for the last eigenvalue not determined if tridiagonalization fails.*

## Protected Member Functions

- void [product](#) (size\_t n, mat\_t &a, vec\_t &w, vec\_t &prod)  
*Naive matrix-vector product.*

### 3.169.2 Member Function Documentation

#### 3.169.2.1 int eigenvalues (size\_t size, mat\_t &mat, size\_t n\_iter, vec\_t &eigen, vec\_t &diag, vec\_t &off\_diag) [inline]

Approximate the largest eigenvalues of a symmetric matrix `mat` using the Lanczos method.

Given a square matrix `mat` with size `size`, this function applies `n_iter` iterations of the Lanczos algorithm to produce `n_iter` approximate eigenvalues stored in `eigen`. As a by-product, this function also partially tridiagonalizes the matrix placing the result in `diag` and `off_diag`. Before calling this function, space must have already been allocated for `eigen`, `diag`, and `off_diag`. All three of these arrays must have at least enough space for `n_iter` elements.

Choosing `n_iter = size` will produce all of the exact eigenvalues and the corresponding tridiagonal matrix, but this may be slower than diagonalizing the matrix directly.

Definition at line 87 of file `lanczos.h`.

#### 3.169.2.2 int eigen\_tdiag (size\_t n, vec\_t &diag, vec\_t &off\_diag) [inline]

In-place diagonalization of a tri-diagonal matrix.

On input, the vectors `diag` and `off_diag` should both be vectors of size `n`. The diagonal entries stored in `diag`, and the  $n - 1$  off-diagonal entries should be stored in `off_diag`, starting with `off_diag[1]`. The value in `off_diag[0]` is unused. The vector `off_diag` is destroyed by the computation.

This uses an implicit QL method from the EISPACK routine `imtql1`. The value of `ierr` from the original Fortran routine is stored in [td\\_lasteval](#).

Definition at line 172 of file `lanczos.h`.

#### 3.169.2.3 void product (size\_t n, mat\_t &a, vec\_t &w, vec\_t &prod) [inline, protected]

Naive matrix-vector product.

It is assumed that memory is already allocated for `prod`.

Definition at line 301 of file `lanczos.h`.

The documentation for this class was generated from the following file:

- `lanczos.h`

## 3.170 lib\_settings\_class Class Reference

```
#include <lib_settings.h>
```

---

### 3.170.1 Detailed Description

A class to manage testing and record success and failure.

Definition at line 39 of file lib\_settings.h.

#### Public Member Functions

- `std::string get_data_dir ()`  
*Return the data directory.*
- `int set_data_dir (std::string dir)`  
*Set the data directory.*
- `std::string get_tmp_dir ()`  
*Return the temp file directory.*
- `int set_tmp_dir (std::string dir)`  
*Set the temp file directory.*

#### Protected Attributes

- `std::string data_dir`  
*The present data directory.*
- `std::string tmp_dir`  
*The present temp file directory.*

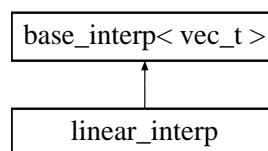
The documentation for this class was generated from the following file:

- [lib\\_settings.h](#)

## 3.171 linear\_interp Class Template Reference

```
#include <interp.h>
```

Inheritance diagram for linear\_interp:



### 3.171.1 Detailed Description

```
template<class vec_t> class linear_interp< vec_t >
```

Linear interpolation (GSL).

Linear interpolation requires no calls to [alloc\(\)](#), [free\(\)](#) or [init\(\)](#), as there is no internal storage required.

Definition at line 132 of file interp.h.

#### Public Member Functions

- [linear\\_interp \(\)](#)



- virtual int [interp](#) (const vec\_t &x\_array, const vec\_t &y\_array, size\_t size, double x, double &y)  
*Give the value of the function  $y(x = x_0)$ .*
- virtual int [deriv](#) (const vec\_t &x\_array, const vec\_t &y\_array, size\_t size, double x, double &dydx)  
*Give the value of the derivative  $y'(x = x_0)$ .*
- virtual int [deriv2](#) (const vec\_t &x, const vec\_t &y, size\_t size, double x0, double &d2ydx2)  
*Give the value of the second derivative  $y''(x = x_0)$ .*
- virtual int [integ](#) (const vec\_t &x\_array, const vec\_t &y\_array, size\_t size, double a, double b, double &result)  
*Give the value of the integral  $\int_a^b y(x) dx$ .*

The documentation for this class was generated from the following file:

- [interp.h](#)

## 3.172 linear\_solver Class Template Reference

```
#include <ode_it_solve.h>
```

### 3.172.1 Detailed Description

**template<class vec\_t, class mat\_t> class linear\_solver< vec\_t, mat\_t >**

A generic solver for the linear system  $Ax = b$ .

Definition at line 124 of file [ode\\_it\\_solve.h](#).

#### Public Member Functions

- virtual [~linear\\_solver](#) ()
- virtual int [solve](#) (size\_t n, mat\_t &a, vec\_t &b, vec\_t &x)

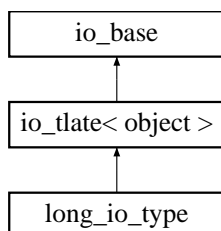
The documentation for this class was generated from the following file:

- [ode\\_it\\_solve.h](#)

## 3.173 long\_io\_type Class Reference

```
#include <collection.h>
```

Inheritance diagram for long\_io\_type::



### 3.173.1 Detailed Description

I/O object for long variables.

Definition at line 1774 of file [collection.h](#).

## Public Member Functions

- [long\\_io\\_type](#) (const char \*t)  
*Desc.*
- [long\\_io\\_type](#) ()
- int [addl](#) ([collection](#) &co, std::string name, unsigned long int x, bool overwrt=true)  
*Add a long to a [collection](#).*
- int [getl](#) ([collection](#) &co, std::string tname)  
*Get a long from a [collection](#).*
- int [get\\_def](#) ([collection](#) &co, std::string tname, unsigned long int &op, unsigned long int def=0)  
*Get a long from a [collection](#).*

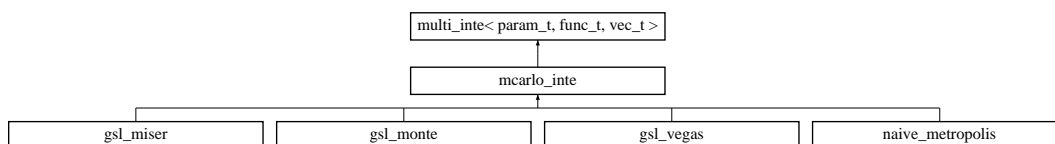
The documentation for this class was generated from the following file:

- collection.h

## 3.174 mcarlo\_inte Class Template Reference

```
#include <mcarlo_inte.h>
```

Inheritance diagram for mcarlo\_inte::



### 3.174.1 Detailed Description

```
template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_view> class mcarlo_inte< param_t,
func_t, rng_t, vec_t >
```

Monte-Carlo integration base.

There is no generic interface here since it would just be the same as that from [multi\\_inte](#). This class just provides the generic Monte Carlo parameters and the random number generator. The default type for the random number generator is a [gsl\\_rnga](#) object.

Definition at line 46 of file mcarlo\_inte.h.

## Public Member Functions

- [mcarlo\\_inte](#) ()
- virtual [~mcarlo\\_inte](#) ()
- virtual const char \* [type](#) ()  
*Return string denoting type ("mcarlo\_inte").*

## Data Fields

- int [n\\_points](#)  
*(default 1000)*
- rng\_t [def\\_rng](#)  
*The random number generator.*

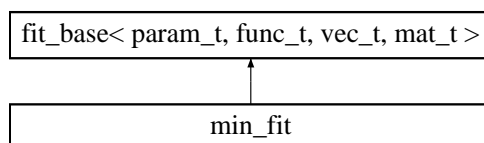
The documentation for this class was generated from the following file:

- mcarlo\_inte.h

## 3.175 min\_fit Class Template Reference

```
#include <min_fit.h>
```

Inheritance diagram for min\_fit::



### 3.175.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t = ovector\_view, class mat\_t = omatrix\_view> class min\_fit< param\_t, func\_t, vec\_t, mat\_t >**

Non-linear least-squares fitting class with generic minimizer.

This minimizes a generic fitting function using any [multi\\_min](#) object, and then uses the GSL routines to calculate the uncertainties in the parameters and the covariance matrix.

This can be useful for fitting problems which might be better handled by more complex minimizers than those that are used in [gsl\\_fit](#). For problems with many local minima near the global minimum, using a [sim\\_anneal](#) object with this class can sometimes produce better results than [gsl\\_fit](#).

Definition at line 53 of file min\_fit.h.

### Public Member Functions

- [min\\_fit](#) ()
- virtual [~min\\_fit](#) ()
- virtual int [fit](#) (size\_t ndat, vec\_t &xdat, vec\_t &ydat, vec\_t &yerr, size\_t npar, vec\_t &par, mat\_t &covar, double &chi2, param\_t &pa, func\_t &fitfun)  
*Fit the data specified in (xdat,ydat) to the function fitfun with the parameters in par.*
- int [set\\_multi\\_min](#) ([multi\\_min](#)< [func\\_par](#) \*, [multi\\_funct](#)< [func\\_par](#) \*, vec\_t > > &mm)  
*Set the [multi\\_min](#) object to use (default is [gsl\\_mmin\\_nmsimplex](#)).*
- virtual const char \* [type](#) ()  
*Return string denoting type ("min\_fit").*

### Data Fields

- [gsl\\_mmin\\_simp](#)< [func\\_par](#) \*, [multi\\_funct](#)< [func\\_par](#) \*, vec\_t > > [def\\_multi\\_min](#)  
*The default minimizer.*

### Protected Member Functions

- double [min\\_func](#) (size\_t np, const vec\_t &xp, [func\\_par](#) \*&fp)  
*The function to [minimize](#).*

## Static Protected Member Functions

- static int [func](#) (const gsl\_vector \*x, void \*pa, gsl\_vector \*f)  
*Evaluate the function.*
- static int [dfunc](#) (const gsl\_vector \*x, void \*pa, gsl\_matrix \*jac)  
*Evaluate the [jacobian](#).*
- static int [fdfunc](#) (const gsl\_vector \*x, void \*pa, gsl\_vector \*f, gsl\_matrix \*jac)  
*Evaluate the function and the [jacobian](#).*

## Protected Attributes

- [multi\\_min](#)< [func\\_par](#) \*, [multi\\_func](#)< [func\\_par](#) \*, vec\_t > > \* [mmp](#)  
*The minimizer.*
- bool [min\\_set](#)  
*True if the minimizer has been set by the user.*

## Data Structures

- struct [func\\_par](#)  
*A structure for passing information to the GSL functions.*

### 3.175.2 Member Function Documentation

**3.175.2.1 virtual int fit (size\_t ndat, vec\_t & xdat, vec\_t & ydat, vec\_t & yerr, size\_t npar, vec\_t & par, mat\_t & covar, double & chi2, param\_t & pa, func\_t & fitfun)** [inline, virtual]

Fit the data specified in (xdat,ydat) to the function fitfun with the parameters in par.

The covariance matrix for the parameters is returned in covar and the value of  $\chi^2$  is returned in chi2.

Reimplemented from [fit\\_base](#).

Definition at line 93 of file min\_fit.h.

The documentation for this class was generated from the following file:

- min\_fit.h

## 3.176 min\_fit::func\_par Struct Reference

```
#include <min_fit.h>
```

### 3.176.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t = ovector\_view, class mat\_t = omatrix\_view> struct min\_fit< param\_t, func\_t, vec\_t, mat\_t >::func\_par**

A structure for passing information to the GSL functions.

This structure is given so that the user can specify the minimizer to use.

Definition at line 70 of file min\_fit.h.

## Data Fields

- `func_t & f`  
*The fitting function.*
- `param_t & vp`  
*The user-specified parameter.*
- `int ndat`  
*The number of data.*
- `vec_t * xdat`  
*The x values.*
- `vec_t * ydat`  
*The y values.*
- `vec_t * yerr`  
*The y uncertainties.*
- `int npar`  
*The number of fitting parameters.*

The documentation for this struct was generated from the following file:

- `min_fit.h`

## 3.177 minimize Class Template Reference

```
#include <minimize.h>
```

### 3.177.1 Detailed Description

`template<class param_t, class func_t, class dfunc_t = func_t> class minimize< param_t, func_t, dfunc_t >`

Numerical differentiation base.

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

## Public Member Functions

- `minimize()`
- `virtual ~minimize()`
- `virtual int print_iter(double x, double y, int iter, double value=0.0, double limit=0.0, std::string comment="")`  
*Print out iteration information.*
- `virtual int min(double &x, double &fmin, param_t &pa, func_t &func)`  
*Calculate the minimum `min` of `func` w.r.t 'x'.*
- `virtual int min_bkt(double &x2, double x1, double x3, double &fmin, param_t &pa, func_t &func)`  
*Calculate the minimum `min` of `func` with `x2` bracketed between `x1` and `x3`.*
- `virtual int min_de(double &x, double &fmin, param_t &pa, func_t &func, dfunc_t &df)`  
*Calculate the minimum `min` of `func` with derivative `dfunc` w.r.t 'x'.*
- `virtual int min_bkt_de(double &x2, double x1, double x3, double &fmin, param_t &pa, func_t &func, dfunc_t &df)`  
*Calculate the minimum `min` of `func` with derivative `dfunc` and `x2` bracketed between `x1` and `x3`.*
- `virtual int bracket(double &ax, double &bx, double &cx, double &fa, double &fb, double &fc, param_t &pa, func_t &func)`  
*Given `ax` and `bx`, bracket a minimum for function `func`.*
- `virtual const char * type()`  
*Return string denoting type ("minimize").*

## Data Fields

- int [verbose](#)  
*Output control.*
- int [ntrial](#)  
*Maximum number of iterations.*
- double [tolf](#)  
*The tolerance for the minimum function value.*
- double [tolx](#)  
*The tolerance for the location of the minimum.*
- int [last\\_ntrial](#)  
*The number of iterations for in the most recent minimization.*

### 3.177.2 Member Function Documentation

**3.177.2.1** `virtual int print_iter (double x, double y, int iter, double value = 0.0, double limit = 0.0, std::string comment = "")` [inline, virtual]

Print out iteration information.

Depending on the value of the variable `verbose`, this prints out the iteration information. If `verbose=0`, then no information is printed, while if `verbose>1`, then after each iteration, the present values of `x` and `y` are output to `std::cout` along with the iteration number. If `verbose>=2` then each iteration waits for a character.

Definition at line 78 of file `minimize.h`.

**3.177.2.2** `virtual int min (double &x, double &fmin, param_t &pa, func_t &func)` [inline, virtual]

Calculate the minimum `min` of `func` w.r.t '`x`'.

If this is not overloaded, it attempts to bracket the minimum using [bracket\(\)](#) and then calls [min\\_bkt\(\)](#) with the newly bracketed minimum.

Definition at line 109 of file `minimize.h`.

**3.177.2.3** `virtual int min_bkt (double &x2, double x1, double x3, double &fmin, param_t &pa, func_t &func)` [inline, virtual]

Calculate the minimum `min` of `func` with `x2` bracketed between `x1` and `x3`.

If this is not overloaded, it ignores the bracket and calls [min\(\)](#).

Reimplemented in [cern\\_minimize](#), and [gsl\\_min\\_brent](#).

Definition at line 122 of file `minimize.h`.

**3.177.2.4** `virtual int min_de (double &x, double &fmin, param_t &pa, func_t &func, dfunc_t &df)` [inline, virtual]

Calculate the minimum `min` of `func` with derivative `dfunc` w.r.t '`x`'.

If this is not overloaded, it attempts to bracket the minimum using [bracket\(\)](#) and then calls [min\\_bkt\\_de\(\)](#) with the newly bracketed minimum.

Definition at line 135 of file `minimize.h`.

**3.177.2.5** `virtual int min_bkt_de (double &x2, double x1, double x3, double &fmin, param_t &pa, func_t &func, dfunc_t &df)` [inline, virtual]

Calculate the minimum `min` of `func` with derivative `dfunc` and `x2` bracketed between `x1` and `x3`.

If this is not overloaded, it ignores the bracket and calls [min\\_de\(\)](#).

Definition at line 150 of file minimize.h.

### 3.177.2.6 virtual int bracket (double & ax, double & bx, double & cx, double & fa, double & fb, double & fc, param\_t & pa, func\_t & func) [inline, virtual]

Given ax and bx, bracket a minimum for function func.

Upon success, fa=f(ax), fb=f(bx), and fc=f(cx) with fb<fa and fb<fc and ax<bx<cx.

#### Todo

Improve this algorithm with the standard golden ratio method.

#### Todo

Double check that this works when at least two of f(a), f(b) and f((a+b)/2) are equal.

Definition at line 167 of file minimize.h.

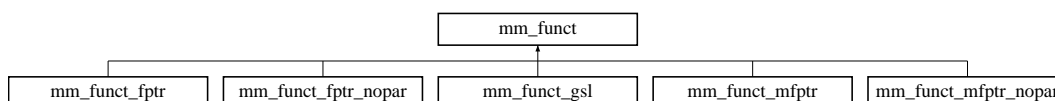
The documentation for this class was generated from the following file:

- [minimize.h](#)

## 3.178 mm\_funct Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_funct::



### 3.178.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view> class mm_funct< param_t, vec_t >
```

Array of multi-dimensional functions.

This class generalizes  $nv$  functions of  $nv$  variables, i.e.  $y_j(x_0, x_1, \dots, x_{nv-1})$  for  $0 \leq j \leq nv - 1$ .

To use C-style arrays, use [mm\\_vfunct](#).

Definition at line 45 of file mm\_funct.h.

#### Public Member Functions

- [mm\\_funct\(\)](#)
- virtual [~mm\\_funct\(\)](#)
- virtual int [operator\(\)](#) (size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa)  
Compute  $nv$  functions,  $y$ , of  $nv$  variables stored in  $x$  with parameter  $pa$ .

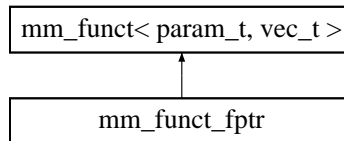
The documentation for this class was generated from the following file:

- [mm\\_funct.h](#)

## 3.179 mm\_funct\_fptr Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_funct\_fptr::



### 3.179.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view> class mm_funct_fptr< param_t, vec_t >
```

Function pointer to array of multi-dimensional functions.

Definition at line 73 of file mm\_funct.h.

#### Public Member Functions

- [mm\\_funct\\_fptr](#) ()
- virtual [~mm\\_funct\\_fptr](#) ()
- [mm\\_funct\\_fptr](#) (int(\*fp)(size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa))  
*Specify the function pointer.*
- int [set\\_function](#) (int(\*fp)(size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa))  
*Specify the function pointer.*
- virtual int [operator\(\)](#) (size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa)  
*Compute nv functions, y, of nv variables stored in x with parameter pa.*

#### Protected Attributes

- int(\* [fptr](#)) (size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa)  
*The function pointer to the user-supplied function.*

#### Private Member Functions

- [mm\\_funct\\_fptr](#) (const [mm\\_funct\\_fptr](#) &)
- [mm\\_funct\\_fptr](#) & [operator=](#) (const [mm\\_funct\\_fptr](#) &)

The documentation for this class was generated from the following file:

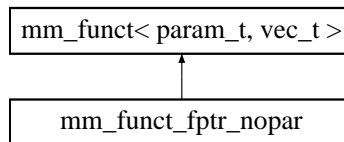
- mm\_funct.h

## 3.180 mm\_funct\_fptr\_nopar Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_funct\_fptr\_nopar::





### 3.180.1 Detailed Description

**template<class param\_t, class vec\_t = ovector\_view> class mm\_funct\_ptr\_nopar< param\_t, vec\_t >**

Function pointer to array of multi-dimensional functions with no parameters.

Definition at line 126 of file mm\_funct.h.

#### Public Member Functions

- [mm\\_funct\\_ptr\\_nopar](#) ()
- virtual [~mm\\_funct\\_ptr\\_nopar](#) ()
- [mm\\_funct\\_ptr\\_nopar](#) (int(\*fp)(size\_t nv, const vec\_t &x, vec\_t &y))  
*Specify the function pointer.*
- int [set\\_function](#) (int(\*fp)(size\_t nv, const vec\_t &x, vec\_t &y))  
*Specify the function pointer.*
- virtual int [operator\(\)](#) (size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa)  
*Compute nv functions, y, of nv variables stored in x with parameter pa.*

#### Protected Attributes

- int(\* [fptr](#) )(size\_t nv, const vec\_t &x, vec\_t &y)  
*The function pointer to the user-supplied function.*

#### Private Member Functions

- [mm\\_funct\\_ptr\\_nopar](#) (const [mm\\_funct\\_ptr\\_nopar](#) &)
- [mm\\_funct\\_ptr\\_nopar](#) & [operator=](#) (const [mm\\_funct\\_ptr\\_nopar](#) &)

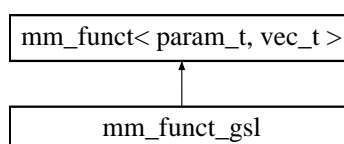
The documentation for this class was generated from the following file:

- mm\_funct.h

## 3.181 mm\_funct\_gsl Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_funct\_gsl::



### 3.181.1 Detailed Description

**template<class param\_t, class vec\_t = ovector\_view> class mm\_funct\_gsl< param\_t, vec\_t >**

Function pointer to a gsl\_multiroot\_function.

This works because with the template parameter `vec_t` as an `ovector_view` class because `ovector_view` is inherited from `gsl_vector`.

Definition at line 181 of file `mm_funct.h`.

#### Public Member Functions

- `mm_funct_gsl` (int(\*fp)(const gsl\_vector \*x, param\_t &pa, gsl\_vector \*f))  
*Specify the function pointer.*
- virtual int `operator()` (size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa)  
*Compute nv functions, y, of nv variables stored in x with parameter pa.*

#### Protected Attributes

- int(\* `fptr`) (const gsl\_vector \*x, param\_t &pa, gsl\_vector \*f)  
*The function pointer to the user-supplied function.*

#### Private Member Functions

- `mm_funct_gsl` (const `mm_funct_gsl` &)
- `mm_funct_gsl` & `operator=` (const `mm_funct_gsl` &)

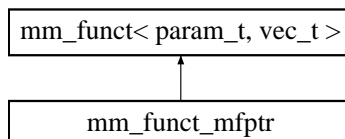
The documentation for this class was generated from the following file:

- `mm_funct.h`

## 3.182 mm\_funct\_mfptr Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for `mm_funct_mfptr`:



### 3.182.1 Detailed Description

**template<class tclass, class param\_t, class vec\_t = ovector\_view> class mm\_funct\_mfptr< tclass, param\_t, vec\_t >**

Member function pointer to an array of multi-dimensional functions.

Definition at line 221 of file `mm_funct.h`.

## Public Member Functions

- [mm\\_funct\\_mfptr](#) ()  
*Empty constructor.*
- [mm\\_funct\\_mfptr](#) (tclass \*tp, int(tclass::\*fp)(size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa))  
*Specify the member function pointer.*
- int [set\\_function](#) (tclass \*tp, int(tclass::\*fp)(size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa))  
*Specify the member function pointer.*
- virtual [~mm\\_funct\\_mfptr](#) ()
- virtual int [operator](#)() (size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa)  
*Compute nv functions, y, of nv variables stored in x with parameter pa.*

## Protected Attributes

- int(tclass::\* [fptr](#))(size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa)  
*The member function pointer.*
- tclass \* [tptr](#)  
*The class pointer.*

## Private Member Functions

- [mm\\_funct\\_mfptr](#) (const [mm\\_funct\\_mfptr](#) &)
- [mm\\_funct\\_mfptr](#) & [operator=](#) (const [mm\\_funct\\_mfptr](#) &)

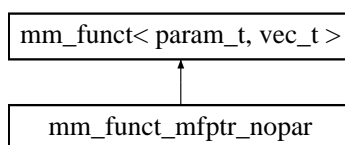
The documentation for this class was generated from the following file:

- mm\_funct.h

## 3.183 mm\_funct\_mfptr\_nopar Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_funct\_mfptr\_nopar::



### 3.183.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_view> class mm_funct_mfptr_nopar< tclass, param_t, vec_t >
```

Member function pointer to an array of multi-dimensional functions.

Definition at line 279 of file mm\_funct.h.

## Public Member Functions

- [mm\\_funct\\_mfptr\\_nopar](#) ()  
*Empty constructor.*
- [mm\\_funct\\_mfptr\\_nopar](#) (tclass \*tp, int(tclass::\*fp)(size\_t nv, const vec\_t &x, vec\_t &y))

*Specify the member function pointer.*

- int [set\\_function](#) (tclass \*tp, int(tclass::\*fp)(size\_t nv, const vec\_t &x, vec\_t &y))

*Specify the member function pointer.*

- virtual [~mm\\_funct\\_mfptr\\_nopar](#) ()
- virtual int [operator\(\)](#) (size\_t nv, const vec\_t &x, vec\_t &y, param\_t &pa)

*Compute  $nv$  functions,  $y$ , of  $nv$  variables stored in  $x$  with parameter  $pa$ .*

### Protected Attributes

- int(tclass::\* [fptr](#)) (size\_t nv, const vec\_t &x, vec\_t &y)

*The member function pointer.*

- tclass \* [tptr](#)

*The class pointer.*

### Private Member Functions

- [mm\\_funct\\_mfptr\\_nopar](#) (const [mm\\_funct\\_mfptr\\_nopar](#) &)
- [mm\\_funct\\_mfptr\\_nopar](#) & [operator=](#) (const [mm\\_funct\\_mfptr\\_nopar](#) &)

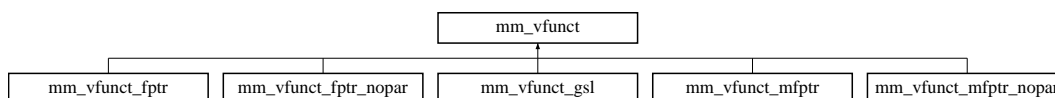
The documentation for this class was generated from the following file:

- [mm\\_funct.h](#)

## 3.184 mm\_vfunct Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_vfunct::



### 3.184.1 Detailed Description

**template<class param\_t, size\_t nv> class mm\_vfunct< param\_t, nv >**

Array of multi-dimensional functions with arrays.

This class generalizes  $nv$  functions of  $nv$  variables, i.e.  $y_j(x_0, x_1, \dots, x_{nv-1})$  for  $0 \leq j \leq nv - 1$ .

To use ovector\_view objects instead of C-style arrays, use [mm\\_funct](#).

Definition at line 345 of file [mm\\_funct.h](#).

### Public Member Functions

- [mm\\_vfunct](#) ()
- virtual [~mm\\_vfunct](#) ()
- virtual int [operator\(\)](#) (size\_t nvar, const double x[nv], double y[nv], param\_t &pa)

*Compute  $nv$  functions,  $y$ , of  $nv$  variables stored in  $x$  with parameter  $pa$ .*

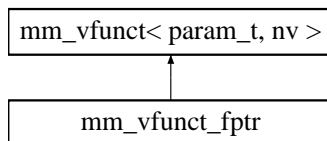
The documentation for this class was generated from the following file:

- [mm\\_funct.h](#)

## 3.185 mm\_vfunct\_fptr Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_vfunct\_fptr::



### 3.185.1 Detailed Description

```
template<class param_t, size_t nv> class mm_vfunct_fptr< param_t, nv >
```

Function pointer to array of multi-dimensional functions with arrays.

Definition at line 374 of file mm\_funct.h.

#### Public Member Functions

- [mm\\_vfunct\\_fptr](#) (int(\*fp)(size\_t nvar, const double x[nv], double y[nv], param\_t &pa))  
*Specify the function pointer.*
- virtual [~mm\\_vfunct\\_fptr](#) ()
- virtual int [operator\(\)](#) (size\_t nvar, const double x[nv], double y[nv], param\_t &pa)  
*Compute nv functions, y, of nv variables stored in x with parameter pa.*

#### Protected Member Functions

- [mm\\_vfunct\\_fptr](#) ()

#### Protected Attributes

- int(\* [fptr](#)) (size\_t nvar, const double x[nv], double y[nv], param\_t &pa)  
*The function pointer.*

#### Private Member Functions

- [mm\\_vfunct\\_fptr](#) (const [mm\\_vfunct\\_fptr](#) &)
- [mm\\_vfunct\\_fptr](#) & [operator=](#) (const [mm\\_vfunct\\_fptr](#) &)

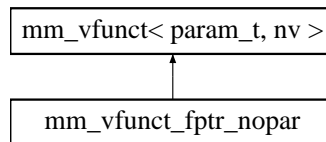
The documentation for this class was generated from the following file:

- mm\_funct.h

## 3.186 mm\_vfunct\_fptr\_nopar Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_vfunct\_fptr\_nopar::



### 3.186.1 Detailed Description

**template<class param\_t, size\_t nv> class mm\_vfunct\_fptr\_nopar< param\_t, nv >**

Function pointer to array of multi-dimensional functions with arrays and no parameters.

Definition at line 420 of file mm\_funct.h.

#### Public Member Functions

- [mm\\_vfunct\\_fptr\\_nopar](#) (int(\*fp)(size\_t nvar, const double x[nv], double y[nv]))  
*Specify the function pointer.*
- virtual [~mm\\_vfunct\\_fptr\\_nopar](#) ()
- virtual int [operator\(\)](#) (size\_t nvar, const double x[nv], double y[nv], param\_t &pa)  
*Compute nv functions, y, of nv variables stored in x with parameter pa.*

#### Protected Member Functions

- [mm\\_vfunct\\_fptr\\_nopar](#) ()

#### Protected Attributes

- int(\* [fptr](#)) (size\_t nvar, const double x[nv], double y[nv])  
*The function pointer.*

#### Private Member Functions

- [mm\\_vfunct\\_fptr\\_nopar](#) (const [mm\\_vfunct\\_fptr\\_nopar](#) &)
- [mm\\_vfunct\\_fptr\\_nopar](#) & [operator=](#) (const [mm\\_vfunct\\_fptr\\_nopar](#) &)

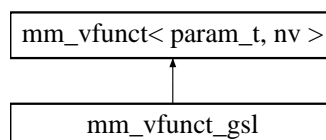
The documentation for this class was generated from the following file:

- mm\_funct.h

## 3.187 mm\_vfunct\_gsl Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_vfunct\_gsl::



### 3.187.1 Detailed Description

**template<class param\_t, size\_t nv> class mm\_vfunct\_gsl< param\_t, nv >**

Function pointer to a gsl\_multirout\_function with arrays.

Definition at line 467 of file mm\_funct.h.

#### Public Member Functions

- [mm\\_vfunct\\_gsl](#) (int(\*fp)(const gsl\_vector \*x, param\_t &pa, gsl\_vector \*f))  
*Specify the function pointer.*
- virtual int [operator\(\)](#) (size\_t nvar, const double x[nv], double y[nv], param\_t &pa)  
*Compute nv functions, y, of nv variables stored in x with parameter pa.*

#### Protected Attributes

- int(\* [fptr](#) )(const gsl\_vector \*x, param\_t &pa, gsl\_vector \*f)  
*The function pointer.*

#### Private Member Functions

- [mm\\_vfunct\\_gsl](#) (const [mm\\_vfunct\\_gsl](#) &)
- [mm\\_vfunct\\_gsl](#) & [operator=](#) (const [mm\\_vfunct\\_gsl](#) &)

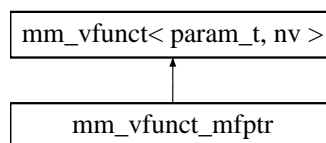
The documentation for this class was generated from the following file:

- mm\_funct.h

## 3.188 mm\_vfunct\_mfpnr Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_vfunct\_mfpnr::



### 3.188.1 Detailed Description

**template<class tclass, class param\_t, size\_t nv> class mm\_vfunct\_mfpnr< tclass, param\_t, nv >**

Member function pointer to an array of multi-dimensional functions with arrays.

Definition at line 508 of file mm\_funct.h.

## Public Member Functions

- [mm\\_vfunct\\_mfptr](#) (tclass \*tp, int(tclass::\*fp)(size\_t nvar, const double x[nv], double y[nv], param\_t &pa))  
*Specify the member function pointer.*
- virtual [~mm\\_vfunct\\_mfptr](#) ()
- virtual int [operator\(\)](#) (size\_t nvar, const double x[nv], double y[nv], param\_t &pa)  
*Compute nv functions, y, of nv variables stored in x with parameter pa.*

## Protected Attributes

- int(tclass::\* [fptr](#)) (size\_t nvar, const double x[nv], double y[nv], param\_t &pa)  
*The member function pointer.*
- tclass \* [tptr](#)  
*The class pointer.*

## Private Member Functions

- [mm\\_vfunct\\_mfptr](#) (const [mm\\_vfunct\\_mfptr](#) &)
- [mm\\_vfunct\\_mfptr](#) & [operator=](#) (const [mm\\_vfunct\\_mfptr](#) &)

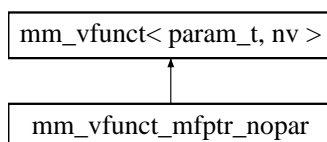
The documentation for this class was generated from the following file:

- mm\_funct.h

## 3.189 mm\_vfunct\_mfptr\_nopar Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm\_vfunct\_mfptr\_nopar::



### 3.189.1 Detailed Description

```
template<class tclass, class param_t, size_t nv> class mm_vfunct_mfptr_nopar< tclass, param_t, nv >
```

Member function pointer to an array of multi-dimensional functions with arrays.

Definition at line 554 of file mm\_funct.h.

## Public Member Functions

- [mm\\_vfunct\\_mfptr\\_nopar](#) (tclass \*tp, int(tclass::\*fp)(size\_t nvar, const double x[nv], double y[nv]))  
*Specify the member function pointer.*
- virtual [~mm\\_vfunct\\_mfptr\\_nopar](#) ()
- virtual int [operator\(\)](#) (size\_t nvar, const double x[nv], double y[nv], param\_t &pa)  
*Compute nv functions, y, of nv variables stored in x with parameter pa.*



## Protected Attributes

- `int(tclass::* fptr)(size_t nvar, const double x[nv], double y[nv])`  
*The member function pointer.*
- `tclass * tptr`  
*The class pointer.*

## Private Member Functions

- `mm_vfunct_mfptr_nopar` (const `mm_vfunct_mfptr_nopar` &)
- `mm_vfunct_mfptr_nopar` & `operator=` (const `mm_vfunct_mfptr_nopar` &)

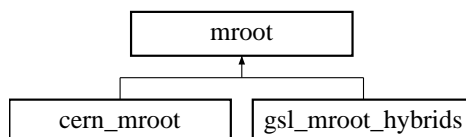
The documentation for this class was generated from the following file:

- `mm_funct.h`

## 3.190 mroot Class Template Reference

```
#include <mroot.h>
```

Inheritance diagram for mroot::



### 3.190.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class jfunc_t = jac_funct<param_t,vec_t,omatrix_view>>
class mroot< param_t, func_t, vec_t, jfunc_t >
```

Multidimensional root-finding base.

**The template parameters:** The template parameter `func_t` specifies the functions to solve and should be a class containing a definition

```
func_t::operator()(size_t nv, const vec_t &x, vec_t &y, param_t &pa);
```

where `y` is the value of the functions at `x` with parameter `pa` and `x` and `y` are array-like classes defining `operator[]` of size `nv`. If the Jacobian matrix is to be specified by the user, then the parameter `jfunc_t` specifies the [jacobian](#) and should contain the definition

```
jfunc_t::operator(size_t nv, vec_t &x, vec_t &y,
mat_t &j, param_t &pa);
```

where `x` is the independent variables, `y` is the array of function values, and `j` is the Jacobian matrix. This template parameter can be ignored if only the function [msolve\(\)](#) will be used.

### Warning:

Many of the routines assume that the scale of the functions and their variables is of order unity. The solution routines may lose precision if this is not the case.

Definition at line 61 of file `mroot.h`.

## Public Member Functions

- [mroot](#) ()
- virtual [~mroot](#) ()
- virtual const char \* [type](#) ()  
*Return the type, "mroot".*
- virtual int [msolve](#) (size\_t n, vec\_t &x, param\_t &pa, func\_t &func)  
*Solve func using x as an initial guess, returning x.*
- virtual int [msolve\\_de](#) (size\_t n, vec\_t &x, param\_t &pa, func\_t &func, jfunc\_t &dfunc)  
*Solve func with derivatives dfunc using x as an initial guess, returning x.*
- template<class vec2\_t, class vec3\_t>  
int [print\\_iter](#) (size\_t n, const vec2\_t &x, const vec3\_t &y, int iter, double value=0.0, double limit=0.0, std::string comment="")  
*Print out iteration information.*

## Data Fields

- double [tolf](#)  
*The maximum value of the functions for success (default 1.0e-8).*
- double [tolx](#)  
*The minimum allowable stepsize (default 1.0e-12).*
- int [verbose](#)  
*Output control (default 0).*
- int [ntrial](#)  
*Maximum number of iterations (default 100).*
- int [last\\_ntrial](#)  
*The number of iterations for in the most recent minimization.*

### 3.190.2 Member Function Documentation

#### 3.190.2.1 virtual int msolve\_de (size\_t n, vec\_t &x, param\_t &pa, func\_t &func, jfunc\_t &dfunc) [inline, virtual]

Solve func with derivatives dfunc using x as an initial guess, returning x.

By default, this function just calls [msolve\(\)](#) and ignores the last argument.

Reimplemented in [gsl\\_mroot\\_hybrids](#).

Definition at line 104 of file mroot.h.

#### 3.190.2.2 int print\_iter (size\_t n, const vec2\_t &x, const vec3\_t &y, int iter, double value = 0.0, double limit = 0.0, std::string comment = "") [inline]

Print out iteration information.

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character.

This is implemented as a template class using a new vector type because sometimes the internal vector class is distinct from the user-specified vector class (e.g. in [gsl\\_mroot\\_hybrids](#)).

Definition at line 124 of file mroot.h.

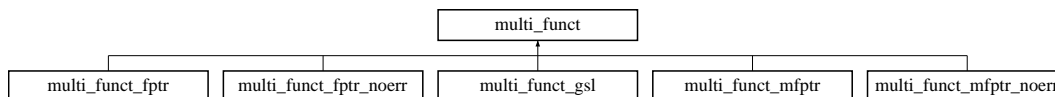
The documentation for this class was generated from the following file:

- mroot.h

## 3.191 multi\_funct Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for multi\_funct::



### 3.191.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view> class multi_funct< param_t, vec_t >
```

Multi-dimensional function base.

This class generalizes one function of several variables, i.e.  $y(x_0, x_1, \dots, x_{nv-1})$  where  $nv$  is the number of variables in the function  $y$ .

Definition at line 41 of file multi\_funct.h.

#### Public Member Functions

- [multi\\_funct\(\)](#)
- virtual [~multi\\_funct\(\)](#)
- virtual int [operator\(\)](#) (size\_t nv, const vec\_t &x, double &y, param\_t &pa)  
*Compute a function  $y$  of  $nv$  variables stored in  $x$  with parameter  $pa$ .*
- virtual double [operator\(\)](#) (size\_t nv, const vec\_t &x, param\_t &pa)  
*Return the value of a function of  $nv$  variables stored in  $x$  with parameter  $pa$ .*

### 3.191.2 Member Function Documentation

#### 3.191.2.1 virtual int operator() (size\_t nv, const vec\_t &x, double &y, param\_t &pa) [inline, virtual]

Compute a function  $y$  of  $nv$  variables stored in  $x$  with parameter  $pa$ .

Compute a function  $y$  of  $nv$  variables stored in  $x$  with parameter  $pa$ .

Reimplemented in [multi\\_funct\\_fptr](#), [multi\\_funct\\_gsl](#), [multi\\_funct\\_fptr\\_noerr](#), [multi\\_funct\\_mfptr](#), and [multi\\_funct\\_mfptr\\_noerr](#).

Definition at line 52 of file multi\_funct.h.

#### 3.191.2.2 virtual double operator() (size\_t nv, const vec\_t &x, param\_t &pa) [inline, virtual]

Return the value of a function of  $nv$  variables stored in  $x$  with parameter  $pa$ .

Note that this is reimplemented in all children because if one member function `operator()` is reimplemented, all must be.

Reimplemented in [multi\\_funct\\_fptr](#), [multi\\_funct\\_gsl](#), [multi\\_funct\\_fptr\\_noerr](#), [multi\\_funct\\_mfptr](#), and [multi\\_funct\\_mfptr\\_noerr](#).

Definition at line 67 of file multi\_funct.h.

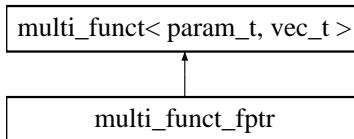
The documentation for this class was generated from the following file:

- [multi\\_funct.h](#)

## 3.192 multi\_funct\_fptr Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for multi\_funct\_fptr::



### 3.192.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view> class multi_funct_fptr< param_t, vec_t >
```

Function pointer to a multi-dimensional function.

Definition at line 87 of file multi\_funct.h.

#### Public Member Functions

- [multi\\_funct\\_fptr](#) (int(\*fp)(size\_t nv, const vec\_t &x, double &y, param\_t &pa))  
*Specify the function pointer.*
- virtual [~multi\\_funct\\_fptr](#) ()
- virtual int [operator\(\)](#) (size\_t nv, const vec\_t &x, double &y, param\_t &pa)  
*Compute a function y of nv variables stored in x with parameter pa.*
- virtual double [operator\(\)](#) (size\_t nv, const vec\_t &x, param\_t &pa)  
*Return the value of a function of nv variables stored in x with parameter pa.*

#### Protected Member Functions

- [multi\\_funct\\_fptr](#) ()

#### Protected Attributes

- int(\* [fptr](#)) (size\_t nv, const vec\_t &x, double &y, param\_t &pa)  
*Store the function pointer.*

#### Private Member Functions

- [multi\\_funct\\_fptr](#) (const [multi\\_funct\\_fptr](#) &)
- [multi\\_funct\\_fptr](#) & [operator=](#) (const [multi\\_funct\\_fptr](#) &)

### 3.192.2 Member Function Documentation

#### 3.192.2.1 virtual double operator() (size\_t nv, const vec\_t &x, param\_t &pa) [inline, virtual]

Return the value of a function of nv variables stored in x with parameter pa.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from [multi\\_funct](#).

Definition at line 114 of file multi\_funct.h.

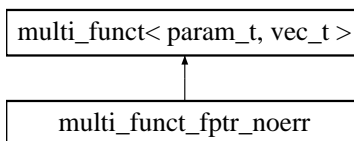
The documentation for this class was generated from the following file:

- multi\_func\_t.h

### 3.193 multi\_func\_t\_fptr\_noerr Class Template Reference

```
#include <multi_func_t.h>
```

Inheritance diagram for multi\_func\_t\_fptr\_noerr::



#### 3.193.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view> class multi_func_t_fptr_noerr< param_t, vec_t >
```

Function pointer to a multi-dimensional function without error control.

Definition at line 205 of file multi\_func\_t.h.

#### Public Member Functions

- [multi\\_func\\_t\\_fptr\\_noerr](#) (double(\*fp)(size\_t nv, const vec\_t &x, param\_t &pa))  
*Specify the function pointer.*
- virtual [~multi\\_func\\_t\\_fptr\\_noerr](#) ()
- virtual int [operator\(\)](#) (size\_t nv, const vec\_t &x, double &y, param\_t &pa)  
*Compute a function y of nv variables stored in x with parameter pa.*
- virtual double [operator\(\)](#) (size\_t nv, const vec\_t &x, param\_t &pa)  
*Return the value of a function of nv variables stored in x with parameter pa.*

#### Protected Member Functions

- [multi\\_func\\_t\\_fptr\\_noerr](#) ()

#### Protected Attributes

- double(\* [fptr](#)) (size\_t nv, const vec\_t &x, param\_t &pa)  
*Store the function pointer.*

#### Private Member Functions

- [multi\\_func\\_t\\_fptr\\_noerr](#) (const [multi\\_func\\_t\\_fptr\\_noerr](#) &)
- [multi\\_func\\_t\\_fptr\\_noerr](#) & [operator=](#) (const [multi\\_func\\_t\\_fptr\\_noerr](#) &)

### 3.193.2 Member Function Documentation

#### 3.193.2.1 virtual double operator() (size\_t nv, const vec\_t &x, param\_t &pa) [inline, virtual]

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function `operator()` is reimplemented, all must be.

Reimplemented from [multi\\_func\\_t](#).

Definition at line 231 of file `multi_func_t.h`.

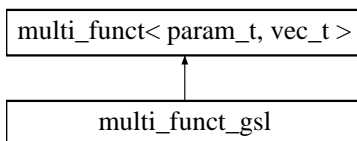
The documentation for this class was generated from the following file:

- `multi_func_t.h`

## 3.194 multi\_func\_t\_gsl Class Template Reference

```
#include <multi_func_t.h>
```

Inheritance diagram for `multi_func_t_gsl`:



### 3.194.1 Detailed Description

```
template<class param_t, class vec_t = ovector_view> class multi_func_t_gsl< param_t, vec_t >
```

Function pointer to a `gsl_multimin_function`.

Definition at line 146 of file `multi_func_t.h`.

#### Public Member Functions

- [multi\\_func\\_t\\_gsl](#) (double(\*fp)(const gsl\_vector \*x, param\_t &pa))  
*Specify the function pointer.*
- virtual [~multi\\_func\\_t\\_gsl](#) ()
- virtual int [operator\(\)](#) (size\_t nv, const vec\_t &x, double &y, param\_t &pa)  
*Compute a function `y` of `nv` variables stored in `x` with parameter `pa`.*
- virtual double [operator\(\)](#) (size\_t nv, const vec\_t &x, param\_t &pa)  
*Return the value of a function of `nv` variables stored in `x` with parameter `pa`.*

#### Protected Member Functions

- [multi\\_func\\_t\\_gsl](#) ()

#### Protected Attributes

- double(\* [fptr](#))(const gsl\_vector \*x, param\_t &pa)  
*Store the function pointer.*

## Private Member Functions

- `multi_funct_gsl` (const `multi_funct_gsl` &)
- `multi_funct_gsl` & `operator=` (const `multi_funct_gsl` &)

### 3.194.2 Member Function Documentation

#### 3.194.2.1 virtual double operator() (size\_t nv, const vec\_t &x, param\_t &pa) [inline, virtual]

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function `operator()` is reimplemented, all must be.

Reimplemented from `multi_funct`.

Definition at line 172 of file `multi_funct.h`.

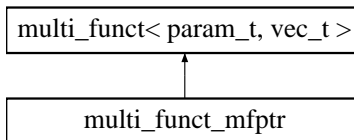
The documentation for this class was generated from the following file:

- `multi_funct.h`

## 3.195 multi\_funct\_mfptr Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for `multi_funct_mfptr`:



### 3.195.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_view> class multi_funct_mfptr< tclass, param_t, vec_t >
```

Member function pointer to a multi-dimensional function.

Definition at line 263 of file `multi_funct.h`.

## Public Member Functions

- `multi_funct_mfptr` (tclass \*tp, int(tclass::\*fp)(size\_t nv, const vec\_t &x, double &y, param\_t &pa))  
*Specify the member function pointer.*
- virtual `~multi_funct_mfptr` ()
- virtual int `operator()` (size\_t nv, const vec\_t &x, double &y, param\_t &pa)  
*Compute a function  $y$  of  $nv$  variables stored in  $x$  with parameter  $pa$ .*
- virtual double `operator()` (size\_t nv, const vec\_t &x, param\_t &pa)  
*Return the value of a function of  $nv$  variables stored in  $x$  with parameter  $pa$ .*

## Protected Attributes

- int(tclass::\* `fptr`)(size\_t nv, const vec\_t &x, double &y, param\_t &pa)  
*Store the function pointer.*
- tclass \* `tptr`  
*Store a pointer to the class instance.*

## Private Member Functions

- **multi\_funct\_mfptr** (const [multi\\_funct\\_mfptr](#) &)
- **multi\_funct\_mfptr & operator=** (const [multi\\_funct\\_mfptr](#) &)

### 3.195.2 Member Function Documentation

#### 3.195.2.1 virtual double operator() (size\_t nv, const vec\_t &x, param\_t &pa) [inline, virtual]

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function `operator()` is reimplemented, all must be.

Reimplemented from [multi\\_funct](#).

Definition at line 289 of file `multi_funct.h`.

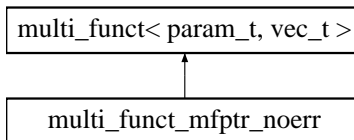
The documentation for this class was generated from the following file:

- `multi_funct.h`

## 3.196 multi\_funct\_mfptr\_noerr Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for `multi_funct_mfptr_noerr`:



### 3.196.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_view> class multi_funct_mfptr_noerr< tclass, param_t, vec_t >
```

Member function pointer to a multi-dimensional function.

Definition at line 319 of file `multi_funct.h`.

## Public Member Functions

- **multi\_funct\_mfptr\_noerr** (tclass \*tp, double(tclass::\*fp)(size\_t nv, const vec\_t &x, param\_t &pa))  
*Specify the member function pointer.*
- virtual **~multi\_funct\_mfptr\_noerr** ()
- virtual int **operator()** (size\_t nv, const vec\_t &x, double &y, param\_t &pa)  
*Compute a function `y` of `nv` variables stored in `x` with parameter `pa`.*
- virtual double **operator()** (size\_t nv, const vec\_t &x, param\_t &pa)  
*Return the value of a function of `nv` variables stored in `x` with parameter `pa`.*

## Protected Attributes

- double(tclass::\* **fptr**)(size\_t nv, const vec\_t &x, param\_t &pa)  
*Store the function pointer.*
- tclass \* **tptr**  
*Store a pointer to the class instance.*



## Private Member Functions

- `multi_funct_mfptr_noerr` (const `multi_funct_mfptr_noerr` &)
- `multi_funct_mfptr_noerr & operator=` (const `multi_funct_mfptr_noerr` &)

## 3.196.2 Member Function Documentation

### 3.196.2.1 virtual double operator() (size\_t nv, const vec\_t &x, param\_t &pa) [inline, virtual]

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function `operator()` is reimplemented, all must be.

Reimplemented from `multi_funct`.

Definition at line 346 of file `multi_funct.h`.

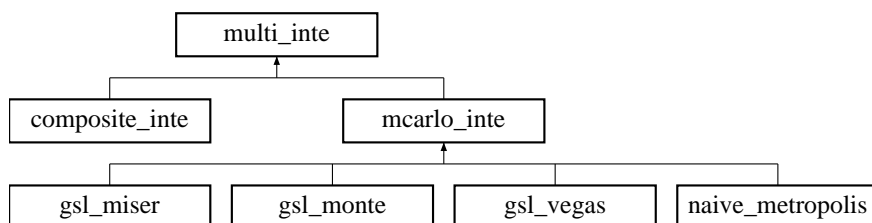
The documentation for this class was generated from the following file:

- `multi_funct.h`

## 3.197 multi\_inte Class Template Reference

```
#include <multi_inte.h>
```

Inheritance diagram for `multi_inte::`



### 3.197.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t = ovector\_view> class multi\_inte< param\_t, func\_t, vec\_t >**

Multi-dimensional integration over a hypercube base class.

Multi-dimensional integration over a region defined by constant limits. For more general regions of integration, use children of the class `gen_inte`.

Definition at line 41 of file `multi_inte.h`.

## Public Member Functions

- `multi_inte` ()
- virtual `~multi_inte` ()
- virtual double `minteg` (func\_t &func, size\_t ndim, const vec\_t &a, const vec\_t &b, param\_t &pa)
- virtual int `minteg_err` (func\_t &func, size\_t ndim, const vec\_t &a, const vec\_t &b, param\_t &pa, double &res, double &err)
- double `get_error` ()  
Return the error in the result from the last call to `minteg()` or `minteg_err()`.
- const char \* `type` ()  
Return string denoting type ("multi\_inte").

## Data Fields

- int [verbose](#)  
Verbosity.
- double [tolf](#)  
The maximum "uncertainty" in the value of the integral (default  $10^{-8}$ ).

## Protected Attributes

- double [interror](#)  
The uncertainty for the last integration computation.

### 3.197.2 Member Function Documentation

#### 3.197.2.1 double get\_error () [inline]

Return the error in the result from the last call to [minteg\(\)](#) or [minteg\\_err\(\)](#).

This will quietly return zero if no integrations have been performed.

Definition at line 94 of file multi\_inte.h.

The documentation for this class was generated from the following file:

- multi\_inte.h

## 3.198 multi\_min Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for multi\_min::



### 3.198.1 Detailed Description

```
template<class param_t, class func_t, class dfunc_t = func_t, class vec_t = ovector_view> class multi_min< param_t, func_t, dfunc_t, vec_t >
```

Multidimensional minimization base.

**The template parameters:** The template parameter `func_t` specifies the function to [minimize](#) and should be a class containing a definition

```
func_t::operator()(size_t nv, const vec_t &x, double &f, param_t &pa);
```

where `f` is the value of the function at `x` with parameter `pa` where `x` is a array-like class defining `operator[]` of size `nv`. The parameter `dfunc_t` (if used) should provide the [gradient](#) with

```
func_t::operator()(size_t nv, const vec_t &x, vec_t &g, param_t &pa);
```

where `g` is the [gradient](#) of the function at `x`.

Definition at line 414 of file multi\_min.h.

## Public Member Functions

- `multi_min()`
- `virtual ~multi_min()`
- `virtual int mmin(size_t nvar, vec_t &x, double &fmin, param_t &pa, func_t &func)`  
Calculate the minimum min of func w.r.t. the array x of size nvar.
- `virtual int mmin_de(size_t nvar, vec_t &x, double &fmin, param_t &pa, func_t &func, dfunc_t &dfunc)`  
Calculate the minimum min of func w.r.t. the array x of size nvar with *gradient* dfunc.
- `template<class vec2_t>`  
`int print_iter(size_t nv, vec2_t &x, double y, int iter, double value, double limit, std::string comment)`  
Print out iteration information.
- `const char * type()`  
Return string denoting type ("multi\_min").

## Data Fields

- `int verbose`  
Output control.
- `int ntrial`  
Maximum number of iterations.
- `double tolf`  
Tolerance.
- `double tolx`  
The minimum allowable stepsize.
- `int last_ntrial`  
The number of iterations for in the most recent minimization.

### 3.198.2 Member Function Documentation

#### 3.198.2.1 `int print_iter(size_t nv, vec2_t &x, double y, int iter, double value, double limit, std::string comment)` [inline]

Print out iteration information.

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character.

Definition at line 473 of file multi\_min.h.

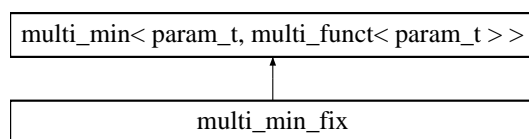
The documentation for this class was generated from the following file:

- multi\_min.h

## 3.199 multi\_min\_fix Class Template Reference

```
#include <multi_min_fix.h>
```

Inheritance diagram for multi\_min\_fix::



### 3.199.1 Detailed Description

**template<class param\_t, class bool\_vec\_t> class multi\_min\_fix< param\_t, bool\_vec\_t >**

Multidimensional minimizer fixing some variables and varying others.

Definition at line 37 of file multi\_min\_fix.h.

#### Public Member Functions

- [multi\\_min\\_fix](#) ()  
*Specify the member function pointer.*
- virtual [~multi\\_min\\_fix](#) ()
- virtual int [mmin](#) (size\_t nvar, [ovector\\_view](#) &x, double &fmin, param\_t &pa, [multi\\_func\\_t](#)< param\_t > &func)  
*Calculate the minimum min of func w.r.t. the array x of size nvar.*
- virtual int [mmin\\_fix](#) (size\_t nvar, [ovector\\_view](#) &x, double &fmin, bool\_vec\_t &fix, param\_t &pa, [multi\\_func\\_t](#)< param\_t > &func)  
*Calculate the minimum of func while fixing some parameters as specified in fix.*
- int [set\\_mmin](#) ([multi\\_min](#)< param\_t, [multi\\_func\\_t\\_mfptr](#)< [multi\\_min\\_fix](#), param\_t > > &min)  
*Change the base minimizer.*

#### Data Fields

- [gsl\\_mmin\\_simp](#)< param\_t, [multi\\_func\\_t\\_mfptr](#)< [multi\\_min\\_fix](#), param\_t > > [def\\_mmin](#)  
*The default base minimizer.*

#### Protected Member Functions

- virtual int [min\\_func](#) (size\_t nv, const [ovector\\_view](#) &x, double &y, param\_t &pa)  
*The new function to send to the minimizer.*

#### Protected Attributes

- [multi\\_min](#)< param\_t, [multi\\_func\\_t\\_mfptr](#)< [multi\\_min\\_fix](#), param\_t > > \* [mmp](#)  
*The minimizer.*
- [multi\\_func\\_t](#)< param\_t > \* [funcp](#)  
*The user-specified function.*
- size\_t [unv](#)  
*The user-specified number of variables.*
- size\_t [nv\\_new](#)  
*The new number of variables.*
- bool\_vec\_t \* [fixp](#)  
*Specify which parameters to fix.*
- [ovector\\_view](#) \* [xp](#)  
*The user-specified initial vector.*

#### Private Member Functions

- [multi\\_min\\_fix](#) (const [multi\\_min\\_fix](#) &)
- [multi\\_min\\_fix](#) & [operator=](#) (const [multi\\_min\\_fix](#) &)

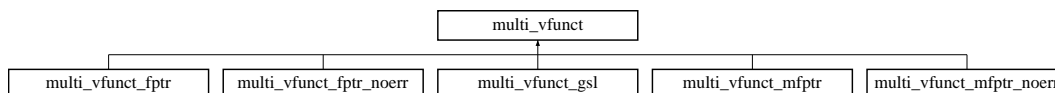
The documentation for this class was generated from the following file:

- multi\_min\_fix.h

## 3.200 multi\_vfunct Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for multi\_vfunct::



### 3.200.1 Detailed Description

```
template<class param_t, size_t nvar> class multi_vfunct< param_t, nvar >
```

Multi-dimensional function base with arrays.

This class generalizes one function of several variables, i.e.  $y(x_0, x_1, \dots, x_{nv-1})$  where  $nv$  is the number of variables in the function  $y$ .

Definition at line 383 of file multi\_funct.h.

#### Public Member Functions

- [multi\\_vfunct\(\)](#)
- virtual [~multi\\_vfunct\(\)](#)
- virtual int [operator\(\)](#) (size\_t nv, const double x[nvar], double &y, param\_t &pa)  
*Compute a function  $y$  of  $nv$  variables stored in  $x$  with parameter  $pa$ .*
- virtual double [operator\(\)](#) (size\_t nv, const double x[nvar], param\_t &pa)  
*Return the value of a function of  $nv$  variables stored in  $x$  with parameter  $pa$ .*

### 3.200.2 Member Function Documentation

#### 3.200.2.1 virtual double operator() (size\_t nv, const double x[nvar], param\_t &pa) [inline, virtual]

Return the value of a function of  $nv$  variables stored in  $x$  with parameter  $pa$ .

Note that this is reimplemented in all children because if one member function `operator()` is reimplemented, all must be.

Reimplemented in [multi\\_vfunct\\_fptr](#), [multi\\_vfunct\\_gsl](#), [multi\\_vfunct\\_fptr\\_noerr](#), [multi\\_vfunct\\_mfptr](#), and [multi\\_vfunct\\_mfptr\\_noerr](#).

Definition at line 405 of file multi\_funct.h.

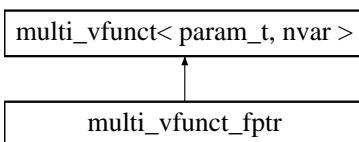
The documentation for this class was generated from the following file:

- multi\_funct.h

## 3.201 multi\_vfunct\_fptr Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for multi\_vfunct\_fptr::



### 3.201.1 Detailed Description

**template<class param\_t, size\_t nvar> class multi\_vfunct\_fptr< param\_t, nvar >**

Function pointer to a multi-dimensional function with arrays.

Definition at line 425 of file multi\_funct.h.

#### Public Member Functions

- [multi\\_vfunct\\_fptr](#) (int(\*fp)(size\_t nv, const double x[nvar], double &y, param\_t &pa))  
*Specify the function pointer.*
- virtual [~multi\\_vfunct\\_fptr](#) ()
- virtual int [operator\(\)](#) (size\_t nv, const double x[nvar], double &y, param\_t &pa)  
*Compute a function y of nv variables stored in x with parameter pa.*
- virtual double [operator\(\)](#) (size\_t nv, const double x[nvar], param\_t &pa)  
*Return the value of a function of nv variables stored in x with parameter pa.*

#### Protected Member Functions

- [multi\\_vfunct\\_fptr](#) ()

#### Protected Attributes

- int(\* [fptr](#)) (size\_t nv, const double x[nvar], double &y, param\_t &pa)  
*Store the function pointer.*

#### Private Member Functions

- [multi\\_vfunct\\_fptr](#) (const [multi\\_vfunct\\_fptr](#) &)
- [multi\\_vfunct\\_fptr](#) & [operator=](#) (const [multi\\_vfunct\\_fptr](#) &)

### 3.201.2 Member Function Documentation

#### 3.201.2.1 virtual double operator() (size\_t nv, const double x[nvar], param\_t &pa) [inline, virtual]

Return the value of a function of nv variables stored in x with parameter pa.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from [multi\\_vfunct](#).

Definition at line 453 of file multi\_funct.h.

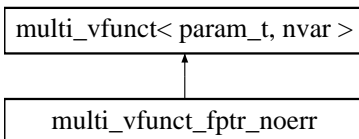
The documentation for this class was generated from the following file:

- multi\_funct.h

## 3.202 multi\_vfunct\_fptr\_noerr Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for multi\_vfunct\_fptr\_noerr::



### 3.202.1 Detailed Description

```
template<class param_t, size_t nvar> class multi_vfunct_fptr_noerr< param_t, nvar >
```

Function pointer to a multi-dimensional function with arrays and without error control.

Definition at line 545 of file multi\_funct.h.

#### Public Member Functions

- [multi\\_vfunct\\_fptr\\_noerr](#) (double(\*fp)(size\_t nv, const double x[nvar], param\_t &pa))  
*Specify the function pointer.*
- virtual [~multi\\_vfunct\\_fptr\\_noerr](#) ()
- virtual int [operator\(\)](#) (size\_t nv, const double x[nvar], double &y, param\_t &pa)  
*Compute a function y of nv variables stored in x with parameter pa.*
- virtual double [operator\(\)](#) (size\_t nv, const double x[nvar], param\_t &pa)  
*Return the value of a function of nv variables stored in x with parameter pa.*

#### Protected Member Functions

- [multi\\_vfunct\\_fptr\\_noerr](#) ()

#### Protected Attributes

- double(\* [fptr](#) )(size\_t nv, const double x[nvar], param\_t &pa)  
*Store the function pointer.*

#### Private Member Functions

- [multi\\_vfunct\\_fptr\\_noerr](#) (const [multi\\_vfunct\\_fptr\\_noerr](#) &)
- [multi\\_vfunct\\_fptr\\_noerr](#) & [operator=](#) (const [multi\\_vfunct\\_fptr\\_noerr](#) &)

### 3.202.2 Member Function Documentation

#### 3.202.2.1 virtual double operator() (size\_t nv, const double x[nvar], param\_t &pa) [inline, virtual]

Return the value of a function of nv variables stored in x with parameter pa.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from [multi\\_vfunct](#).

Definition at line 572 of file multi\_funct.h.

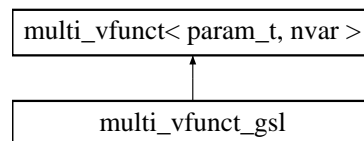
The documentation for this class was generated from the following file:

- multi\_funct.h

### 3.203 multi\_vfunct\_gsl Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for multi\_vfunct\_gsl::



#### 3.203.1 Detailed Description

```
template<class param_t, size_t nvar> class multi_vfunct_gsl< param_t, nvar >
```

Function pointer to a gsl\_multimin\_function with arrays.

Definition at line 485 of file multi\_funct.h.

#### Public Member Functions

- [multi\\_vfunct\\_gsl](#) (double(\*fp)(const gsl\_vector \*x, param\_t &pa))  
*Specify the function pointer.*
- virtual [~multi\\_vfunct\\_gsl](#) ()
- virtual int [operator\(\)](#) (size\_t nv, const double x[nvar], double &y, param\_t &pa)  
*Compute a function y of nv variables stored in x with parameter pa.*
- virtual double [operator\(\)](#) (size\_t nv, const double x[nvar], param\_t &pa)  
*Return the value of a function of nv variables stored in x with parameter pa.*

#### Protected Member Functions

- [multi\\_vfunct\\_gsl](#) ()

#### Protected Attributes

- double(\* [fptr](#) )(const gsl\_vector \*x, param\_t &pa)  
*Store the function pointer.*

#### Private Member Functions

- [multi\\_vfunct\\_gsl](#) (const [multi\\_vfunct\\_gsl](#) &)
- [multi\\_vfunct\\_gsl](#) & [operator=](#) (const [multi\\_vfunct\\_gsl](#) &)



### 3.203.2 Member Function Documentation

#### 3.203.2.1 virtual double operator() (size\_t nv, const double x[nvar], param\_t &pa) [inline, virtual]

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function `operator()` is reimplemented, all must be.

Reimplemented from [multi\\_vfunct](#).

Definition at line 512 of file `multi_funct.h`.

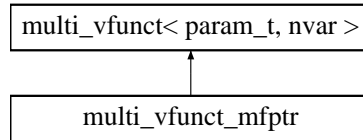
The documentation for this class was generated from the following file:

- `multi_funct.h`

## 3.204 multi\_vfunct\_mfptr Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for `multi_vfunct_mfptr`:



### 3.204.1 Detailed Description

```
template<class tclass, class param_t, size_t nvar> class multi_vfunct_mfptr< tclass, param_t, nvar >
```

Member function pointer to a multi-dimensional function with arrays.

Definition at line 605 of file `multi_funct.h`.

#### Public Member Functions

- [multi\\_vfunct\\_mfptr](#) (tclass \*tp, int(tclass::\*fp)(size\_t nv, const double x[nvar], double &y, param\_t &pa))  
*Specify the member function pointer.*
- virtual [~multi\\_vfunct\\_mfptr](#) ()
- virtual int [operator\(\)](#) (size\_t nv, const double x[nvar], double &y, param\_t &pa)  
*Compute a function `y` of `nv` variables stored in `x` with parameter `pa`.*
- virtual double [operator\(\)](#) (size\_t nv, const double x[nvar], param\_t &pa)  
*Return the value of a function of `nv` variables stored in `x` with parameter `pa`.*

#### Protected Attributes

- int(tclass::\* [fptr](#)) (size\_t nv, const double x[nvar], double &y, param\_t &pa)  
*Store the function pointer.*
- tclass \* [tptr](#)  
*Store a pointer to the class instance.*

## Private Member Functions

- **multi\_vfunct\_mfptr** (const [multi\\_vfunct\\_mfptr](#) &)
- **multi\_vfunct\_mfptr** & **operator=** (const [multi\\_vfunct\\_mfptr](#) &)

### 3.204.2 Member Function Documentation

#### 3.204.2.1 virtual double operator() (size\_t nv, const double x[nvar], param\_t &pa) [inline, virtual]

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function `operator()` is reimplemented, all must be.

Reimplemented from [multi\\_vfunct](#).

Definition at line 633 of file `multi_funct.h`.

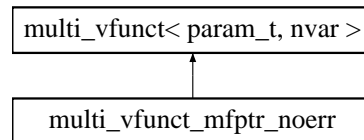
The documentation for this class was generated from the following file:

- `multi_funct.h`

## 3.205 multi\_vfunct\_mfptr\_noerr Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for `multi_vfunct_mfptr_noerr`:



### 3.205.1 Detailed Description

```
template<class tclass, class param_t, size_t nvar> class multi_vfunct_mfptr_noerr< tclass, param_t, nvar >
```

Member function pointer to a multi-dimensional function with arrays.

Definition at line 666 of file `multi_funct.h`.

## Public Member Functions

- **multi\_vfunct\_mfptr\_noerr** (tclass \*tp, double(tclass::\*fp)(size\_t nv, const double x[nvar], param\_t &pa))  
*Specify the member function pointer.*
- virtual **~multi\_vfunct\_mfptr\_noerr** ()
- virtual int **operator()** (size\_t nv, const double x[nvar], double &y, param\_t &pa)  
*Compute a function  $y$  of `nv` variables stored in `x` with parameter `pa`.*
- virtual double **operator()** (size\_t nv, const double x[nvar], param\_t &pa)  
*Return the value of a function of `nv` variables stored in `x` with parameter `pa`.*

## Protected Attributes

- double(tclass::\* **fptr**) (size\_t nv, const double x[nvar], param\_t &pa)  
*Store the function pointer.*
- tclass \* **tptr**  
*Store a pointer to the class instance.*

## Private Member Functions

- **multi\_vfunct\_mfptr\_noerr** (const [multi\\_vfunct\\_mfptr\\_noerr](#) &)
- **multi\_vfunct\_mfptr\_noerr & operator=** (const [multi\\_vfunct\\_mfptr\\_noerr](#) &)

### 3.205.2 Member Function Documentation

#### 3.205.2.1 virtual double operator() (size\_t nv, const double x[nvar], param\_t & pa) [inline, virtual]

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function `operator()` is reimplemented, all must be.

Reimplemented from [multi\\_vfunct](#).

Definition at line 694 of file `multi_funct.h`.

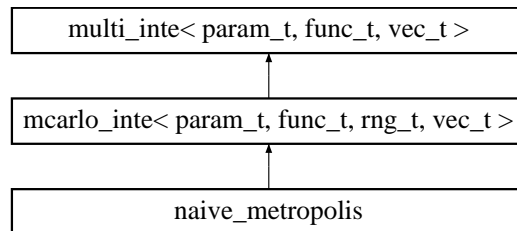
The documentation for this class was generated from the following file:

- `multi_funct.h`

## 3.206 naive\_metropolis Class Template Reference

```
#include <naive_metropolis.h>
```

Inheritance diagram for `naive_metropolis`::



### 3.206.1 Detailed Description

**template<class param\_t, class func\_t, class rng\_t, class vec\_t, class alloc\_vec\_t> class naive\_metropolis< param\_t, func\_t, rng\_t, vec\_t, alloc\_vec\_t >**

Naive metropolis monte carlo integration.

Very experimental.

This returns the ratio of n-dimensional integrals

$$I = \frac{\int f(\vec{x}) \exp[-\beta w(\vec{x})] d\vec{x}}{\int \exp[-\beta w(\vec{x})] d\vec{x}}$$

for function  $f$  specified in `func` and  $w$  specified in `weight`.

A "Metropolis step" (or simply a "step") is defined by the following procedure: The step  $\vec{x}_0 \rightarrow \vec{x}_1$  is accepted if

$$w(\vec{x}_1) < w(\vec{x}_0)$$

or if

$$e^{-\beta[w(\vec{x}_1) - w(\vec{x}_0)]} > r$$

where  $r$  is a uniform random number  $[0, 1)$  newly generated for each step.

`nstart` steps are taken initially to ensure the first point is created with the correct probability. Afterwards, the average value of the function is stored over `niter` steps. This average value is computed `nblock` times to get an overall average and an estimate of the uncertainty.

#### Todo

Talk about how accurate this is with  $\beta$ .

#### Todo

Redo statistics, talk about how many times the integral is evaluated.

#### Todo

Offer an option of giving more results than just the final average and error?

### Minimization

There is a connection between integration over functions of this type and minimization. The expression

$$\lim_{T \rightarrow 0} \frac{\int_{x=a}^{x=b} f(x) e^{-f(x)/T}}{\int_{x=a}^{x=b} e^{-f(x)/T}}$$

gives the minimum of the function  $f(x)$  in the region  $x \in (a, b)$ .

Definition at line 88 of file naive\_metropolis.h.

### Public Member Functions

- `naive_metropolis()`
- virtual `~naive_metropolis()`
- virtual int `minteg_err` (func\_t &func, func\_t &energy, size\_t ndim, const vec\_t &a, const vec\_t &b, const double beta, double &value, double &err, param\_t &pa)  
*Calculate the integral returning result in value.*
- virtual int `minteg_array` (func\_t &func, func\_t &energy, size\_t ndim, const vec\_t &a, const vec\_t &b, const double beta, vec\_t &results, param\_t &pa)  
*Calculate the integral returning result in value.*

### Data Fields

- `rng_t rng`
- unsigned long int `niter`  
*The number of iterations (default 10000).*
- unsigned long int `nstart`  
*The number of warm-up iterations (default 1000).*
- unsigned long int `nblock`  
*Number of blocks (default 10).*

## 3.206.2 Member Function Documentation

**3.206.2.1** `virtual int minteg_err (func_t &func, func_t &energy, size_t ndim, const vec_t &a, const vec_t &b, const double beta, double &value, double &err, param_t &pa) [inline, virtual]`

Calculate the integral returning result in value.

Calculates the integral given the functions `func` and `weight` over the region specified by `a` and `b`, both of side `ndim`. The value of  $\beta$  is specified in `beta`, and the result is returned in `value`, with the uncertainty given in `err`.

Definition at line 113 of file naive\_metropolis.h.

**3.206.2.2** `virtual int minteg_array (func_t &func, func_t &energy, size_t ndim, const vec_t &a, const vec_t &b, const double beta, vec_t &results, param_t &pa) [inline, virtual]`

Calculate the integral returning result in `value`.

Calculates the integral given the functions `func` and `weight` over the region specified by `a` and `b`, both of side `ndim`. The value of  $\beta$  is specified in `beta`, and the result is returned in `value`, with the uncertainty given in `err`.

Definition at line 192 of file `naive_metropolis.h`.

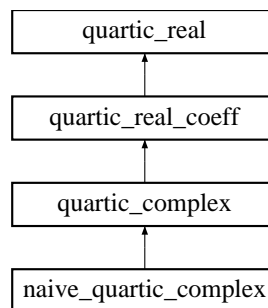
The documentation for this class was generated from the following file:

- `naive_metropolis.h`

## 3.207 naive\_quartic\_complex Class Reference

```
#include <poly.h>
```

Inheritance diagram for `naive_quartic_complex`:



### 3.207.1 Detailed Description

Solve a quartic with complex coefficients and complex roots.

Definition at line 655 of file `poly.h`.

#### Public Member Functions

- `virtual ~naive_quartic_complex ()`
- `virtual int solve_c (const std::complex< double > a4, const std::complex< double > b4, const std::complex< double > c4, const std::complex< double > d4, const std::complex< double > e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)`
- `const char * type ()`  
*Return a string denoting the type ("naive\_quartic\_complex").*

### 3.207.2 Member Function Documentation

**3.207.2.1** `virtual int solve_c (const std::complex< double > a4, const std::complex< double > b4, const std::complex< double > c4, const std::complex< double > d4, const std::complex< double > e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4) [virtual]`

Solves the complex polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four complex solutions  $x = x_1, x = x_2, x = x_3$ , and  $x = x_4$ .

Reimplemented from `quartic_complex`.

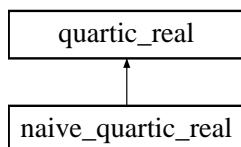
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.208 naive\_quartic\_real Class Reference

```
#include <poly.h>
```

Inheritance diagram for naive\_quartic\_real::



### 3.208.1 Detailed Description

Solve a quartic with real coefficients and real roots.

#### Todo

3/8/07 - Compilation at the NSCL produced non-finite values in [solve\\_r\(\)](#) for some values of the coefficients. This should be checked.

#### Todo

It looks like this code is tested only for a4=1, and if so, the tests should be generalized.

#### Todo

Also, there is a hard-coded number in here ( $10^{-6}$ ), which might be moved to a data member?

Definition at line 638 of file poly.h.

### Public Member Functions

- virtual [~naive\\_quartic\\_real](#) ()
- virtual int [solve\\_r](#) (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)
- const char \* [type](#) ()  
*Return a string denoting the type ("naive\_quartic\_real").*

### 3.208.2 Member Function Documentation

**3.208.2.1** virtual int [solve\\_r](#) (const double *a4*, const double *b4*, const double *c4*, const double *d4*, const double *e4*, double &*x1*, double &*x2*, double &*x3*, double &*x4*) [virtual]

Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented from [quartic\\_real](#).

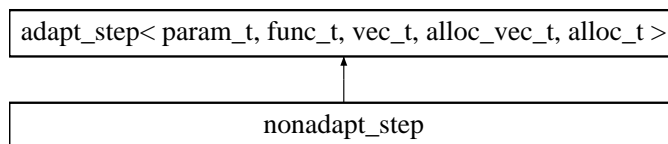
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.209 nonadapt\_step Class Template Reference

```
#include <nonadapt_step.h>
```

Inheritance diagram for nonadapt\_step::



### 3.209.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>
class nonadapt_step< param_t, func_t, vec_t, alloc_vec_t, alloc_t >
```

An non-adaptive stepper implementation of [adapt\\_step](#).

Definition at line 39 of file nonadapt\_step.h.

#### Public Member Functions

- [nonadapt\\_step](#) ()
- virtual [~nonadapt\\_step](#) ()
- virtual int [astep](#) (double &x, double &h, double xmax, size\_t n, vec\_t &y, param\_t &pa, func\_t &derivs)  
*Make an adaptive integration step of the system derivs.*
- virtual int [astep\\_derivs](#) (double &x, double &h, double xmax, size\_t n, vec\_t &y, vec\_t &dydx, param\_t &pa, func\_t &derivs)  
*Make an adaptive integration step of the system derivs.*

#### Protected Attributes

- alloc\_t [ao](#)  
*Memory allocator for objects of type alloc\_vec\_t.*

### 3.209.2 Member Function Documentation

**3.209.2.1** virtual int [astep](#) (double & x, double & h, double xmax, size\_t n, vec\_t & y, param\_t & pa, func\_t & derivs)  
[inline, virtual]

Make an adaptive integration step of the system derivs.

This attempts to take a step of size h from the point x of an n-dimensional system derivs starting with y. On exit, x and y contain the new values at the end of the step and h contains the size of the step.

Reimplemented from [adapt\\_step](#).

Definition at line 57 of file nonadapt\_step.h.

**3.209.2.2** virtual int [astep\\_derivs](#) (double & x, double & h, double xmax, size\_t n, vec\_t & y, vec\_t & dydx, param\_t & pa, func\_t & derivs) [inline, virtual]

Make an adaptive integration step of the system derivs.

This attempts to take a step of size  $h$  from the point  $x$  of an  $n$ -dimensional system `derivs` starting with  $y$ . On exit,  $x$  and  $y$  contain the new values at the end of the step and  $h$  contains the size of the step.

Reimplemented from [adapt\\_step](#).

Definition at line 83 of file `nonadapt_step.h`.

The documentation for this class was generated from the following file:

- `nonadapt_step.h`

## 3.210 o2scl\_interp Class Template Reference

```
#include <interp.h>
```

### 3.210.1 Detailed Description

```
template<class vec_t = ovector_view, class rvec_t = ovector_const_reverse> class o2scl_interp< vec_t, rvec_t >
```

Interpolation class.

This interpolation class is intended to be sufficiently general to handle any vector type. Interpolation of [ovector](#) like objects is performed with the default template parameters, and [array\\_interp](#) is provided for simple interpolation on C-style arrays.

The class automatically handles decreasing arrays by converting from an object of type `vec_t` to an object of type `rvec_t`.

While `vec_t` may be any vector type which allows indexing via `[]`, `rvec_t` must be a vector type which allows indexing and has a constructor with one of the two forms

```
rvec_t::rvec_t(vec_t &v);
rvec_t::rvec_t(vec_t v);
```

so that [o2scl\\_interp](#) can automatically "reverse" a vector if necessary.

It is important that different instances of [o2scl\\_interp\\_vec](#) and [o2scl\\_interp](#) not be given the same interpolation objects, as they will clash.

Definition at line 673 of file `interp.h`.

### Public Member Functions

- [o2scl\\_interp](#) ([base\\_interp](#)< `vec_t` > &`it`, [base\\_interp](#)< `rvec_t` > &`rit`)  
*Create with base interpolation objects `it` and `rit`.*
- [o2scl\\_interp](#) ([base\\_interp](#)< `vec_t` > &`it`)  
*Create with base interpolation object `it` and use [def\\_ritp](#) for reverse interpolation if necessary.*
- [o2scl\\_interp](#) ()  
*Create an interpolator using [def\\_itp](#) and [def\\_ritp](#).*
- virtual [~o2scl\\_interp](#) ()
- virtual double [interp](#) (const double `x0`, size\_t `n`, const `vec_t` &`x`, const `vec_t` &`y`)  
*Give the value of the function  $y(x = x_0)$ .*
- virtual double [deriv](#) (const double `x0`, size\_t `n`, const `vec_t` &`x`, const `vec_t` &`y`)  
*Give the value of the derivative  $y'(x = x_0)$ .*
- virtual double [deriv2](#) (const double `x0`, size\_t `n`, const `vec_t` &`x`, const `vec_t` &`y`)  
*Give the value of the second derivative  $y''(x = x_0)$ .*
- virtual double [integ](#) (const double `x1`, const double `x2`, size\_t `n`, const `vec_t` &`x`, const `vec_t` &`y`)  
*Give the value of the integral  $\int_a^b y(x) dx$ .*
- int [set\\_type](#) ([base\\_interp](#)< `vec_t` > &`it`, [base\\_interp](#)< `rvec_t` > &`rit`)  
*Set base interpolation object.*



## Data Fields

- [cspline\\_interp](#)< vec\_t > [def\\_itp](#)  
*Default base interpolation object (cubic spline with natural boundary conditions).*
- [cspline\\_interp](#)< rvec\_t > [def\\_ritp](#)  
*Default base interpolation object for reversed vectors (cubic spline with natural boundary conditions).*

## Protected Attributes

- [base\\_interp](#)< vec\_t > \* [itp](#)  
*Pointer to base interpolation object.*
- [base\\_interp](#)< rvec\_t > \* [ritp](#)  
*Pointer to base interpolation object for reversed vectors.*

The documentation for this class was generated from the following file:

- [interp.h](#)

## 3.211 o2scl\_interp\_vec Class Template Reference

```
#include <interp.h>
```

### 3.211.1 Detailed Description

**template<class vec\_t = ovector\_view, class alloc\_vec\_t = ovector, class alloc\_t = ovector\_alloc> class o2scl\_interp\_vec< vec\_t, alloc\_vec\_t, alloc\_t >**

Interpolation class for pre-specified vector.

This interpolation class is intended to be sufficiently general to handle any vector type. Interpolation of [ovector](#) like objects is performed with the default template parameters, and [array\\_interp\\_vec](#) is provided for simple interpolation on C-style arrays.

The class automatically handles decreasing arrays by copying the old array to a reversed version. For several interpolations on the same data, copying the initial array can be faster than overloading operator[ ].

It is important that different instances of [o2scl\\_interp\\_vec](#) and [o2scl\\_interp](#) not be given the same interpolation objects, as they will clash.

### Todo

Need to fix constructor to behave properly if init() fails. It should free the memory and set `ln` to zero.

Definition at line 945 of file [interp.h](#).

## Public Member Functions

- [o2scl\\_interp\\_vec](#) ([base\\_interp](#)< vec\_t > &it, size\_t n, const vec\_t &x, const vec\_t &y)  
*Create with base interpolation object it.*
- virtual [~o2scl\\_interp\\_vec](#) ()
- virtual double [interp](#) (const double x0)  
*Give the value of the function  $y(x = x_0)$ .*
- virtual double [deriv](#) (const double x0)  
*Give the value of the derivative  $y'(x = x_0)$ .*
- virtual double [deriv2](#) (const double x0)  
*Give the value of the second derivative  $y''(x = x_0)$ .*

- virtual double `integ` (const double x1, const double x2)  
*Give the value of the integral  $\int_a^b y(x) dx$ .*
- int `set_type` (`base_interp`< `vec_t` > &it)  
*Set base interpolation object.*

## Data Fields

- `cspline_interp`< `vec_t` > `def_itp`  
*Default base interpolation object (cubic spline with natural boundary conditions).*

## Protected Attributes

- `alloc_t ao`  
*Memory allocator for objects of type `alloc_vec_t`.*
- `base_interp`< `vec_t` > \* `itp`  
*Pointer to base interpolation object.*
- bool `inc`  
*True if the original array was increasing.*
- const `vec_t` \* `lx`  
*Pointer to the user-specified x vector.*
- const `vec_t` \* `ly`  
*Pointer to the user-specified x vector.*
- `alloc_vec_t lrx`  
*Reversed version of x.*
- `alloc_vec_t lry`  
*Reversed version of y.*
- `size_t ln`  
*Size of user-specified vectors.*

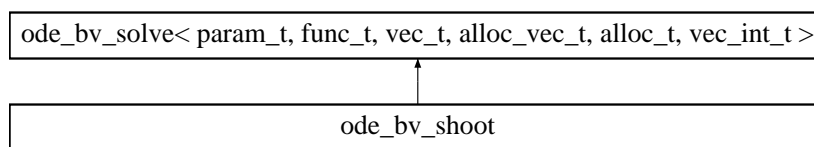
The documentation for this class was generated from the following file:

- `interp.h`

## 3.212 ode\_bv\_shoot Class Template Reference

```
#include <ode_bv_solve.h>
```

Inheritance diagram for `ode_bv_shoot`:



### 3.212.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc,
class vec_int_t = ovector_int_view> class ode_bv_shoot< param_t, func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >
```

Solve boundary-value ODE problems by shooting.

Definition at line 146 of file `ode_bv_solve.h`.

## Public Member Functions

- [ode\\_bv\\_shoot](#) ()
- virtual [~ode\\_bv\\_shoot](#) ()
- virtual int [solve](#) (double x0, double x1, double h, size\_t n, vec\_t &ystart, vec\_t &yend, vec\_int\_t &index, param\_t &pa, func\_t &derivs)  
*Solve the boundary-value problem.*

## Protected Member Functions

- int [solve\\_fun](#) (size\_t nv, const vec\_t &sx, vec\_t &sy, param\_t &pa)

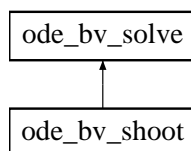
The documentation for this class was generated from the following file:

- [ode\\_bv\\_solve.h](#)

## 3.213 ode\_bv\_solve Class Template Reference

```
#include <ode_bv_solve.h>
```

Inheritance diagram for `ode_bv_solve`:



### 3.213.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc,
class vec_int_t = ovector_int_view> class ode_bv_solve< param_t, func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >
```

Solve boundary-value ODE problems.

Definition at line 43 of file `ode_bv_solve.h`.

## Public Member Functions

- [ode\\_bv\\_solve](#) ()
- virtual [~ode\\_bv\\_solve](#) ()
- virtual int [solve](#) (double x0, double x1, double h, size\_t n, vec\_t &ystart, vec\_t &yend, vec\_int\_t &index, param\_t &pa, func\_t &derivs)  
*Solve the boundary-value problem.*
- int [set\\_iv](#) ([ode\\_iv\\_solve](#)< param\_t, func\_t, vec\_t, alloc\_vec\_t, alloc\_t > &ois)  
*Set initial value solver.*
- int [set\\_mroot](#) ([mroot](#)< param\_t, [mm\\_funct](#)< param\_t > > &root)  
*Set the equation solver.*

## Data Fields

- `ode_iv_solve`< param\_t, func\_t, vec\_t, alloc\_vec\_t, alloc\_t > `def_ois`  
*The default initial value solver.*
- `gsl_mroot_hybrids`< param\_t, `mm_func_t`< param\_t > > `def_mroot`  
*The default equation solver.*
- `int verbose`  
*Set output level.*

## Static Public Attributes

### Values for the index array

- static const `int unk` = 0  
*Unknown on both the left and right boundaries.*
- static const `int right` = 1  
*Known on the right boundary.*
- static const `int left` = 2  
*Known on the left boundary.*
- static const `int both` = 3  
*Known on both the left and right boundaries.*

## Protected Attributes

- `ode_iv_solve`< param\_t, func\_t, vec\_t, alloc\_vec\_t, alloc\_t > \* `ois`  
*The solver for the initial value problem.*
- `mroot`< param\_t, `mm_func_t`< param\_t > > \* `mrootp`  
*The equation solver.*
- `vec_int_t` \* `l_index`  
*The index defining the boundary conditions.*
- `vec_t` \* `l_ystart`  
*Storage for the starting vector.*
- `vec_t` \* `l_yend`  
*Storage for the ending vector.*
- `double l_x0`  
*Storage for the starting point.*
- `double l_x1`  
*Storage for the ending abscissa.*
- `double l_h`  
*Storage for the stepsize.*
- `func_t` \* `l_derivs`  
*The functions to integrate.*
- `size_t l_n`  
*The number of functions.*

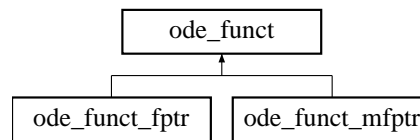
The documentation for this class was generated from the following file:

- `ode_bv_solve.h`

## 3.214 ode\_func Class Template Reference

```
#include <ode_func.h>
```

Inheritance diagram for `ode_func`::



### 3.214.1 Detailed Description

**template<class param\_t, class vec\_t = ovector\_view> class ode\_funct< param\_t, vec\_t >**

Ordinary differential equation function base.

Definition at line 37 of file ode\_funct.h.

#### Public Member Functions

- [ode\\_funct](#) ()
- virtual [~ode\\_funct](#) ()
- virtual int [operator](#)() (double x, size\_t nv, const vec\_t &y, vec\_t &dydx, param\_t &pa)  
*The overloaded operator().*

#### Private Member Functions

- [ode\\_funct](#) (const [ode\\_funct](#) &)
- [ode\\_funct](#) & [operator=](#) (const [ode\\_funct](#) &)

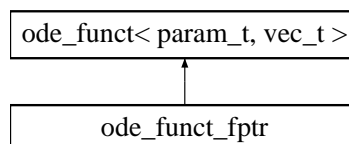
The documentation for this class was generated from the following file:

- ode\_funct.h

## 3.215 ode\_funct\_fptr Class Template Reference

```
#include <ode_funct.h>
```

Inheritance diagram for ode\_funct\_fptr::



### 3.215.1 Detailed Description

**template<class param\_t, class vec\_t = ovector\_view> class ode\_funct\_fptr< param\_t, vec\_t >**

Function pointer to a function.

Definition at line 62 of file ode\_funct.h.

## Public Member Functions

- `ode_funct_fptr` (int(\*fp)(double, size\_t, const vec\_t &, vec\_t &, param\_t &))  
*Specify the function pointer.*
- virtual `~ode_funct_fptr` ()
- virtual int `operator()` (double x, size\_t nv, const vec\_t &y, vec\_t &dydx, param\_t &pa)  
*The overloaded operator().*

## Protected Member Functions

- `ode_funct_fptr` ()

## Protected Attributes

- int(\* `fptr`) (double x, size\_t nv, const vec\_t &y, vec\_t &dydx, param\_t &)  
*The function pointer.*

## Private Member Functions

- `ode_funct_fptr` (const `ode_funct_fptr` &)
- `ode_funct_fptr` & `operator=` (const `ode_funct_fptr` &)

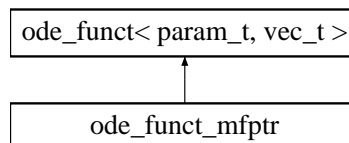
The documentation for this class was generated from the following file:

- `ode_funct.h`

## 3.216 ode\_funct\_mfptr Class Template Reference

```
#include <ode_funct.h>
```

Inheritance diagram for `ode_funct_mfptr`:



### 3.216.1 Detailed Description

**template<class tclass, class param\_t, class vec\_t = ovector\_view> class ode\_funct\_mfptr< tclass, param\_t, vec\_t >**

Member function pointer to a one-dimensional function.

Definition at line 103 of file `ode_funct.h`.

## Public Member Functions

- `ode_funct_mfptr` (tclass \*tp, int(tclass::\*fp)(double x, size\_t nv, const vec\_t &y, vec\_t &dydx, param\_t &))  
*Specify the member function pointer.*
- virtual `~ode_funct_mfptr` ()
- virtual int `operator()` (double x, size\_t nv, const vec\_t &y, vec\_t &dydx, param\_t &pa)  
*The overloaded operator().*

### Protected Attributes

- `int(tclass::* fptr)(double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &)`  
*The pointer to the member function.*
- `tclass * tptr`  
*The pointer to the class.*

### Private Member Functions

- `ode_func_t_mfptr` (const `ode_func_t_mfptr` &)
- `ode_func_t_mfptr` & `operator=` (const `ode_func_t_mfptr` &)

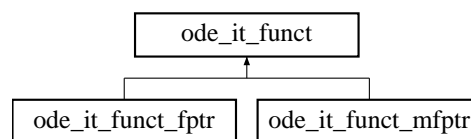
The documentation for this class was generated from the following file:

- `ode_func.h`

## 3.217 ode\_it\_func Class Template Reference

```
#include <ode_it_solve.h>
```

Inheritance diagram for `ode_it_func`:



### 3.217.1 Detailed Description

```
template<class vec_t = o2scl::ovector_view> class ode_it_func< vec_t >
```

Function class for `ode_it_solve`.

Definition at line 49 of file `ode_it_solve.h`.

### Public Member Functions

- `ode_it_func` ()
- virtual `~ode_it_func` ()
- virtual double `operator()` (size\_t ieq, double x, vec\_t &y)  
*Using  $x$  and  $y$ , return the value of function number  $ieq$ .*

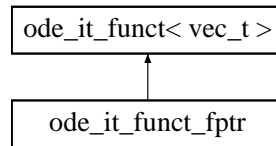
The documentation for this class was generated from the following file:

- `ode_it_solve.h`

## 3.218 ode\_it\_func\_fptr Class Template Reference

```
#include <ode_it_solve.h>
```

Inheritance diagram for `ode_it_func_fptr`:



### 3.218.1 Detailed Description

**template<class vec\_t = o2scl::ovector\_view> class ode\_it\_funct\_fptr< vec\_t >**

Function pointer for [ode\\_it\\_solve](#).

Definition at line 66 of file ode\_it\_solve.h.

#### Public Member Functions

- virtual [~ode\\_it\\_funct\\_fptr](#) ()
- [ode\\_it\\_funct\\_fptr](#) (double(\*fp)(size\_t, double, vec\_t &))  
*Create using a function pointer.*
- virtual double [operator\(\)](#) (size\_t ieq, double x, vec\_t &y)  
*Using x and y, return the value of function number ieq.*

#### Protected Member Functions

- [ode\\_it\\_funct\\_fptr](#) ()

#### Protected Attributes

- double(\* [fptr](#))(size\_t ieq, double x, vec\_t &y)  
*The function pointer.*

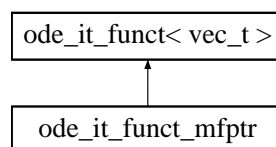
The documentation for this class was generated from the following file:

- ode\_it\_solve.h

## 3.219 ode\_it\_funct\_mfptr Class Template Reference

```
#include <ode_it_solve.h>
```

Inheritance diagram for ode\_it\_funct\_mfptr::



### 3.219.1 Detailed Description

**template<class tclass, class vec\_t = o2scl::ovector\_view> class ode\_it\_funct\_mfptr< tclass, vec\_t >**

Member function pointer for [ode\\_it\\_solve](#).



Definition at line 92 of file ode\_it\_solve.h.

### Public Member Functions

- virtual [~ode\\_it\\_func\\_t\\_mfptr](#) ()
- [ode\\_it\\_func\\_t\\_mfptr](#) (tclass \*tp, double(tclass::\*fp)(size\_t, double, vec\_t &))  
*Create using a class instance and member function.*
- virtual double [operator\(\)](#) (size\_t ieq, double x, vec\_t &y)  
*Using x and y, return the value of function number ieq.*

### Protected Member Functions

- [ode\\_it\\_func\\_t\\_mfptr](#) ()

### Protected Attributes

- tclass \* [tptr](#)  
*The class pointer.*
- double(tclass::\* [fptr](#) )(size\_t ieq, double x, vec\_t &y)  
*The member function pointer.*

The documentation for this class was generated from the following file:

- ode\_it\_solve.h

## 3.220 ode\_it\_make\_Coord Class Reference

```
#include <ode_it_solve.h>
```

### 3.220.1 Detailed Description

Make a coordinate matrix for [ode\\_it\\_solve](#).

Definition at line 197 of file ode\_it\_solve.h.

### Public Member Functions

- o2scl::Coord\_Mat \* [make](#) (size\_t ngrid, size\_t neq, size\_t nbleft)  
*Create a compressed-column format matrix for [ode\\_it\\_solve](#).*

### Data Fields

- o2scl::uvector\_int \* [r](#)  
*The row index.*
- o2scl::uvector\_int \* [c](#)  
*The column pointer.*
- o2scl::uvector \* [vals](#)  
*The matrix entries.*

The documentation for this class was generated from the following file:

- ode\_it\_solve.h
-

## 3.221 ode\_it\_solve Class Template Reference

```
#include <ode_it_solve.h>
```

### 3.221.1 Detailed Description

`template<class func_t, class vec_t, class mat_t, class matrix_row_t, class solver_vec_t, class solver_mat_t> class ode_it_solve< func_t, vec_t, mat_t, matrix_row_t, solver_vec_t, solver_mat_t >`

ODE solver using a generic linear solver to solve finite-difference equations.

#### Todo

Max and average tolerance?

#### Todo

partial correction option?

Definition at line 270 of file ode\_it\_solve.h.

### Public Member Functions

- `ode_it_solve()`
- `virtual ~ode_it_solve()`
- `int set_solver(linear_solver< solver_vec_t, solver_mat_t > &ls)`  
*Set the linear solver.*
- `int solve(size_t ngrid, size_t neq, size_t nbleft, vec_t &x, mat_t &y, func_t &derivs, func_t &left, func_t &right, solver_mat_t &mat, solver_vec_t &rhs, solver_vec_t &dy)`  
*Solve derivs with boundary conditions left and right.*

### Data Fields

- `int verbose`  
*Set level of output (default 0).*
- `double h`  
*Stepsize for finite differencing (default  $10^{-4}$ ).*
- `double tolf`  
*Tolerance (default  $10^{-8}$ ).*
- `size_t niter`  
*Maximum number of iterations (default 30).*

### Protected Member Functions

- `double fd_left(size_t ieq, size_t ivar, double x, vec_t &y)`  
*Compute the derivatives of the LHS boundary conditions.*
- `double fd_right(size_t ieq, size_t ivar, double x, vec_t &y)`  
*Compute the derivatives of the RHS boundary conditions.*
- `double fd_derivs(size_t ieq, size_t ivar, double x, vec_t &y)`  
*Compute the finite-differenced part of the differential equations.*

## Protected Attributes

- `linear_solver`< solver\_vec\_t, solver\_mat\_t > \* `solver`  
*Solver.*

## Storage for functions

- `ode_it_func_t`< vec\_t > \* `fl`
- `ode_it_func_t`< vec\_t > \* `fr`
- `ode_it_func_t`< vec\_t > \* `fd`

The documentation for this class was generated from the following file:

- `ode_it_solve.h`

## 3.222 ode\_iv\_solve Class Template Reference

```
#include <ode_iv_solve.h>
```

### 3.222.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc,
class adapt_step = gsl_astep<param_t,func_t,vec_t,alloc_vec_t,alloc_t>> class ode_iv_solve< param_t, func_t, vec_t, alloc_
vec_t, alloc_t, adapt_step >
```

Base for solving initial-value ODE problems.

#### Todo

Consider modifying so that this can handle tables which are too small by removing `o2sc1f` the rows and doubling the stepsize.

#### Todo

Decide what to do if the adaptive stepper fails. (1/18/07)

- two approaches: continue no matter what, or just stop)

#### Todo

Convert to using `astep_derivs()`?

Definition at line 49 of file `ode_iv_solve.h`.

## Public Member Functions

- `ode_iv_solve()`
- virtual `~ode_iv_solve()`
- `template<class mat_t>`  
int `solve_table` (double x0, double x1, double h, size\_t n, vec\_t &ystart, size\_t &nsol, vec\_t &xsol, mat\_t &ysol, param\_t &pa, func\_t &derivs)  
*Solve the initial-value problem and output a [table](#).*
- `template<class mat_t>`  
int `solve_grid` (double x0, double x1, double h, size\_t n, vec\_t &ystart, size\_t nsol, vec\_t &xsol, mat\_t &ysol, param\_t &pa, func\_t &derivs)  
*Solve the initial-value problem from x0 to x1 over a grid.*
- int `solve_final_value` (double x0, double x1, double h, size\_t n, vec\_t &ystart, vec\_t &yend, param\_t &pa, func\_t &derivs)  
*Solve the initial-value problem to get the final value.*

## Data Fields

- int [verbose](#)  
*Set output level.*
- [adapt\\_step](#) *astepper*  
*The adaptive stepper utilized.*

## Protected Member Functions

- virtual int [print\\_iter](#) (double x, size\_t nv, vec\_t &y)  
*Print out iteration information.*

### 3.222.2 Member Function Documentation

**3.222.2.1** `int solve_table (double x0, double x1, double h, size_t n, vec_t & ystart, size_t & nsol, vec_t & xsol, mat_t & ysol, param_t & pa, func_t & derivs) [inline]`

Solve the initial-value problem and output a [table](#).

Initially, `xsol` should be a vector of size `nsol`, and `ysol` should be a two-dimensional array (i.e. `omatrix_view`) of size `[nsol][n]`. On exit, `nsol` will be the size of the solution [table](#), less than or equal to the original value of `nsol`.

Definition at line 70 of file `ode_iv_solve.h`.

**3.222.2.2** `int solve_grid (double x0, double x1, double h, size_t n, vec_t & ystart, size_t nsol, vec_t & xsol, mat_t & ysol, param_t & pa, func_t & derivs) [inline]`

Solve the initial-value problem from `x0` to `x1` over a grid.

Initially, `xsol` should be an array of size `nsol`, and `ysol` should be a `omatrix` of size `[nsol][n]`.

This function never takes a step larger than the grid size. This could cause inaccuracy if the grid size is too fine.

Definition at line 115 of file `ode_iv_solve.h`.

**3.222.2.3** `int solve_final_value (double x0, double x1, double h, size_t n, vec_t & ystart, vec_t & yend, param_t & pa, func_t & derivs) [inline]`

Solve the initial-value problem to get the final value.

For a particular adaptive stepper, this will likely be the fastest approach, but it provides no information about the solution other than the final value at `x=x1`.

Definition at line 152 of file `ode_iv_solve.h`.

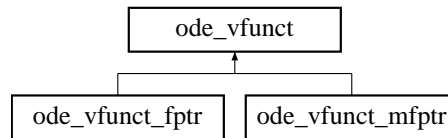
The documentation for this class was generated from the following file:

- `ode_iv_solve.h`

## 3.223 ode\_vfunct Class Template Reference

```
#include <ode_funct.h>
```

Inheritance diagram for `ode_vfunct`:



### 3.223.1 Detailed Description

**template<class param\_t, size\_t nv> class ode\_vfunct< param\_t, nv >**

Ordinary differential equation function base.

Definition at line 150 of file ode\_func.h.

#### Public Member Functions

- [ode\\_vfunct\(\)](#)
- virtual [~ode\\_vfunct\(\)](#)
- virtual int [operator\(\)](#) (double x, size\_t nvar, const double y[nv], double dydx[nv], param\_t &pa)  
The overloaded operator().

#### Private Member Functions

- **ode\_vfunct** (const [ode\\_vfunct](#) &)
- [ode\\_vfunct](#) & **operator=** (const [ode\\_vfunct](#) &)

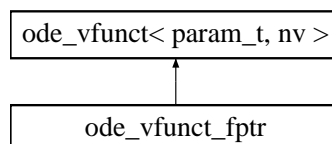
The documentation for this class was generated from the following file:

- ode\_func.h

## 3.224 ode\_vfunct\_fptr Class Template Reference

```
#include <ode_func.h>
```

Inheritance diagram for ode\_vfunct\_fptr::



### 3.224.1 Detailed Description

**template<class param\_t, size\_t nv> class ode\_vfunct\_fptr< param\_t, nv >**

Function pointer to a function.

Definition at line 175 of file ode\_func.h.

## Public Member Functions

- `ode_vfunct_fptr` (int(\*fp)(double, size\_t, const double y[nv], double dydx[nv], param\_t &))  
*Specify the function pointer.*
- virtual `~ode_vfunct_fptr` ()
- virtual int `operator()` (double x, size\_t nvar, const double y[nv], double dydx[nv], param\_t &pa)  
*The overloaded operator().*

## Protected Member Functions

- `ode_vfunct_fptr` ()

## Protected Attributes

- int(\* `fptr`) (double x, size\_t nvar, const double y[nv], double dydx[nv], param\_t &)  
*The function pointer.*

## Private Member Functions

- `ode_vfunct_fptr` (const `ode_vfunct_fptr` &)
- `ode_vfunct_fptr` & `operator=` (const `ode_vfunct_fptr` &)

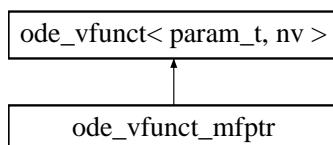
The documentation for this class was generated from the following file:

- `ode_funct.h`

## 3.225 ode\_vfunct\_mfptr Class Template Reference

```
#include <ode_funct.h>
```

Inheritance diagram for `ode_vfunct_mfptr`:



### 3.225.1 Detailed Description

```
template<class tclass, class param_t, size_t nv> class ode_vfunct_mfptr< tclass, param_t, nv >
```

Member function pointer to a one-dimensional function.

Definition at line 216 of file `ode_funct.h`.

## Public Member Functions

- `ode_vfunct_mfptr` (tclass \*tp, int(tclass::\*fp)(double x, size\_t nvar, const double y[nv], double dydx[nv], param\_t &))  
*Specify the member function pointer.*
- virtual `~ode_vfunct_mfptr` ()
- virtual int `operator()` (double x, size\_t nvar, const double y[nv], double dydx[nv], param\_t &pa)  
*The overloaded operator().*

**Protected Attributes**

- `int(tclass::* fptr)(double x, size_t nvar, const double y[nv], double dydx[nv], param_t &)`  
*Pointer to the member function.*
- `tclass * tptr`  
*Pointer to the class.*

**Private Member Functions**

- `ode\_vfunct\_mfptr (const ode\_vfunct\_mfptr &)`
- `ode\_vfunct\_mfptr & operator= (const ode\_vfunct\_mfptr &)`

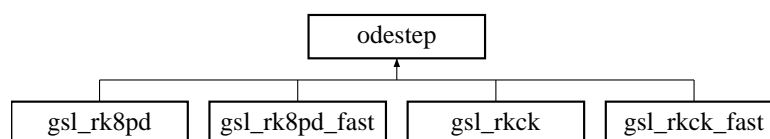
The documentation for this class was generated from the following file:

- `ode_funct.h`

**3.226 odestep Class Template Reference**

```
#include <odestep.h>
```

Inheritance diagram for `odestep::`

**3.226.1 Detailed Description**

```
template<class param_t, class func_t, class vec_t = ovector_view> class odestep< param_t, func_t, vec_t >
```

ODE stepper base.

Definition at line 36 of file `odestep.h`.

**Public Member Functions**

- `odestep ()`
- `virtual ~odestep ()`
- `virtual int get\_order ()`  
*Return the order of the ODE stepper.*
- `virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, param_t &pa, func_t &derivs)`  
*Perform an integration step.*

**Protected Attributes**

- `int order`

### 3.226.2 Member Function Documentation

**3.226.2.1** `virtual int step (double x, double h, size_t n, vec_t & y, vec_t & dydx, vec_t & yout, vec_t & yerr, vec_t & dydx_out, param_t & pa, func_t & derivs)` [`inline`, `virtual`]

Perform an integration step.

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must generally be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` (at `x+h`) using function `derivs` to calculate derivatives. Implementations which do not calculate `yerr` and/or `dydx_out` do not reference these variables so that a blank `vec_t` can be given. All of the implementations allow `yout=y` and `dydx_out=dydx` if necessary.

Reimplemented in [gsl\\_rk8pd](#), [gsl\\_rk8pd\\_fast](#), [gsl\\_rkck](#), and [gsl\\_rkck\\_fast](#).

Definition at line 73 of file `odestep.h`.

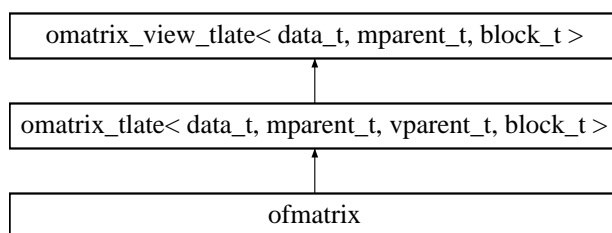
The documentation for this class was generated from the following file:

- `odestep.h`

## 3.227 ofmatrix Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `ofmatrix`::



### 3.227.1 Detailed Description

`template<size_t N, size_t M> class ofmatrix< N, M >`

A matrix where the memory allocation is performed in the constructor.

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

#### Public Member Functions

- [ofmatrix](#) ()

The documentation for this class was generated from the following file:

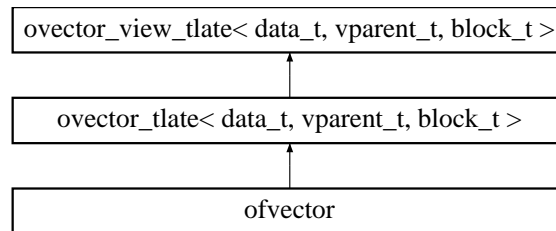
- [omatrix\\_tlate.h](#)

## 3.228 ofvector Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ofvector`::





### 3.228.1 Detailed Description

**template<size\_t N = 0> class ofvector< N >**

A vector where the memory allocation is performed in the constructor.

Definition at line 1874 of file ovector\_tlate.h.

#### Public Member Functions

- [ofvector\(\)](#)

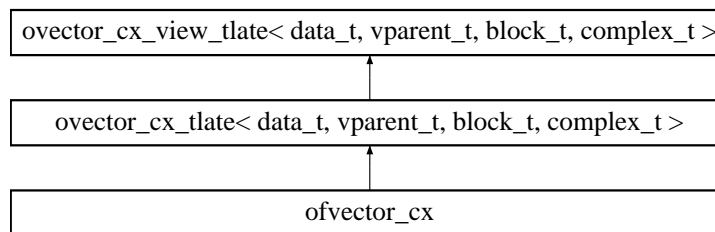
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.229 ofvector\_cx Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ofvector\_cx::



### 3.229.1 Detailed Description

**template<size\_t N = 0> class ofvector\_cx< N >**

A vector where the memory allocation is performed in the constructor.

Definition at line 995 of file ovector\_cx\_tlate.h.

#### Public Member Functions

- [ofvector\\_cx\(\)](#)

The documentation for this class was generated from the following file:

- [ovector\\_cx\\_tlate.h](#)

## 3.230 `omatrix_alloc` Class Reference

```
#include <omatrix_tlate.h>
```

### 3.230.1 Detailed Description

A simple class to provide an `allocate()` function for `omatrix`.

Definition at line 867 of file `omatrix_tlate.h`.

#### Public Member Functions

- void `allocate` (`omatrix` &o, int i, int j)  
*Allocate  $\forall$  for  $i$  elements.*
- void `free` (`omatrix` &o)  
*Free memory.*

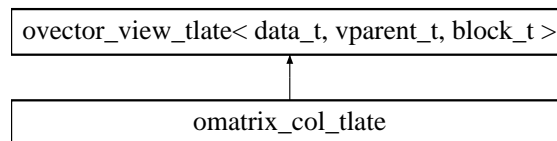
The documentation for this class was generated from the following file:

- [omatrix\\_tlate.h](#)

## 3.231 `omatrix_col_tlate` Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_col_tlate`:



### 3.231.1 Detailed Description

```
template<class data_t, class mparent_t, class vparent_t, class block_t> class omatrix_col_tlate< data_t, mparent_t, vparent_t, block_t >
```

Create a vector from a column of a matrix.

Definition at line 714 of file `omatrix_tlate.h`.

#### Public Member Functions

- `omatrix_col_tlate` (`omatrix_view_tlate`< data\_t, mparent\_t, block\_t > &m, size\_t i)  
*Create a vector from `col i` of matrix m.*

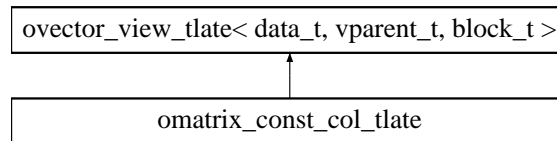
The documentation for this class was generated from the following file:

- [omatrix\\_tlate.h](#)

### 3.232 `omatrix_const_col_tlate` Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_const_col_tlate`:



#### 3.232.1 Detailed Description

**template**<class `data_t`, class `mparent_t`, class `vparent_t`, class `block_t`> class `omatrix_const_col_tlate`< `data_t`, `mparent_t`, `vparent_t`, `block_t` >

Create a const vector from a column of a matrix.

Definition at line 739 of file `omatrix_tlate.h`.

#### Public Member Functions

- [`omatrix\_const\_col\_tlate`](#) ([`omatrix\_view\_tlate`](#)< `data_t`, `mparent_t`, `block_t` > &`m`, `size_t` `i`)  
Create a vector from *col i* of matrix `m`.

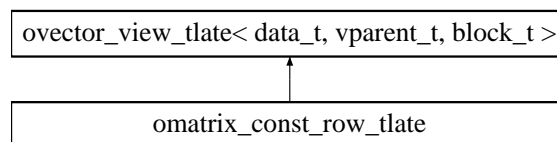
The documentation for this class was generated from the following file:

- [omatrix\\_tlate.h](#)

### 3.233 `omatrix_const_row_tlate` Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_const_row_tlate`:



#### 3.233.1 Detailed Description

**template**<class `data_t`, class `mparent_t`, class `vparent_t`, class `block_t`> class `omatrix_const_row_tlate`< `data_t`, `mparent_t`, `vparent_t`, `block_t` >

Create a const vector from a row of a matrix.

Definition at line 688 of file `omatrix_tlate.h`.

## Public Member Functions

- [`omatrix\_const\_row\_tlate`](#) (`const omatrix\_view\_tlate< data_t, mparent_t, block_t > &m, size_t i`)  
*Create a vector from row `i` of matrix `m`.*

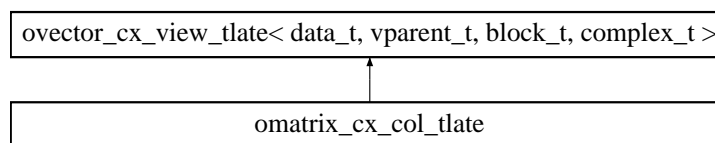
The documentation for this class was generated from the following file:

- [omatrix\\_tlate.h](#)

## 3.234 `omatrix_cx_col_tlate` Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for `omatrix_cx_col_tlate`:



### 3.234.1 Detailed Description

**template<class data\_t, class mparent\_t, class vparent\_t, class block\_t, class complex\_t> class `omatrix_cx_col_tlate`< data\_t, mparent\_t, vparent\_t, block\_t, complex\_t >**

Create a vector from a column of a matrix.

Definition at line 650 of file `omatrix_cx_tlate.h`.

## Public Member Functions

- [`omatrix\_cx\_col\_tlate`](#) (`omatrix\_cx\_view\_tlate< data_t, mparent_t, block_t, complex_t > &m, size_t i`)  
*Create a vector from *col* `i` of matrix `m`.*

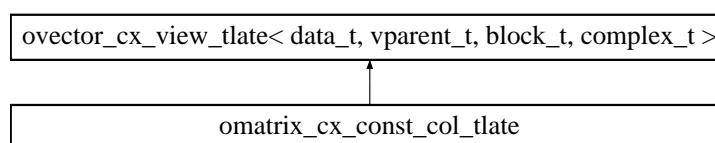
The documentation for this class was generated from the following file:

- [omatrix\\_cx\\_tlate.h](#)

## 3.235 `omatrix_cx_const_col_tlate` Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for `omatrix_cx_const_col_tlate`:



### 3.235.1 Detailed Description

`template<class data_t, class mparent_t, class vparent_t, class block_t, class complex_t> class omatrix_cx_const_col_tlate<data_t, mparent_t, vparent_t, block_t, complex_t >`

Create a vector from a column of a matrix.

Definition at line 670 of file `omatrix_cx_tlate.h`.

#### Public Member Functions

- [`omatrix\_cx\_const\_col\_tlate`](#) ([`omatrix\_cx\_view\_tlate`](#)< data\_t, mparent\_t, block\_t, complex\_t > &m, size\_t i)  
*Create a vector from [col i](#) of matrix m.*

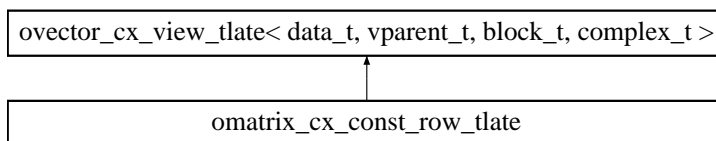
The documentation for this class was generated from the following file:

- [omatrix\\_cx\\_tlate.h](#)

## 3.236 `omatrix_cx_const_row_tlate` Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for `omatrix_cx_const_row_tlate`:



### 3.236.1 Detailed Description

`template<class data_t, class mparent_t, class vparent_t, class block_t, class complex_t> class omatrix_cx_const_row_tlate<data_t, mparent_t, vparent_t, block_t, complex_t >`

Create a vector from a row of a matrix.

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

#### Public Member Functions

- [`omatrix\_cx\_const\_row\_tlate`](#) (const [`omatrix\_cx\_view\_tlate`](#)< data\_t, mparent\_t, block\_t, complex\_t > &m, size\_t i)  
*Create a vector from row i of matrix m.*

The documentation for this class was generated from the following file:

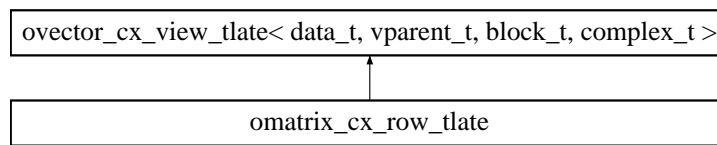
- [omatrix\\_cx\\_tlate.h](#)

## 3.237 `omatrix_cx_row_tlate` Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

---

Inheritance diagram for `omatrix_cx_row_tlate`::



### 3.237.1 Detailed Description

**template<class data\_t, class mparent\_t, class vparent\_t, class block\_t, class complex\_t> class `omatrix_cx_row_tlate`< data\_t, mparent\_t, vparent\_t, block\_t, complex\_t >**

Create a vector from a row of a matrix.

Definition at line 610 of file `omatrix_cx_tlate.h`.

#### Public Member Functions

- [`omatrix\_cx\_row\_tlate`](#) ([`omatrix\_cx\_view\_tlate`](#)< data\_t, mparent\_t, block\_t, complex\_t > &m, size\_t i)  
*Create a vector from row i of matrix m.*

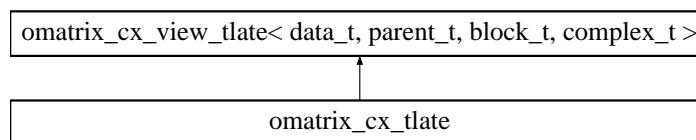
The documentation for this class was generated from the following file:

- [omatrix\\_cx\\_tlate.h](#)

## 3.238 `omatrix_cx_tlate` Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for `omatrix_cx_tlate`::



### 3.238.1 Detailed Description

**template<class data\_t, class parent\_t, class block\_t, class complex\_t> class `omatrix_cx_tlate`< data\_t, parent\_t, block\_t, complex\_t >**

A matrix of double-precision numbers.

Definition at line 384 of file `omatrix_cx_tlate.h`.

#### Public Member Functions

- [`~omatrix\_cx\_tlate`](#) ()

**Standard constructor**

- [omatrix\\_cx\\_tlate](#) (size\_t r=0, size\_t c=0)  
*Create an omatrix of size n with owner as 'true'.*

### Copy constructors

- [omatrix\\_cx\\_tlate](#) (const [omatrix\\_cx\\_tlate](#) &v)  
*Deep copy constructor, allocate new space and make a copy.*
- [omatrix\\_cx\\_tlate](#) (const [omatrix\\_cx\\_view\\_tlate](#)< data\_t, parent\_t, block\_t, complex\_t > &v)  
*Deep copy constructor, allocate new space and make a copy.*

### Memory allocation

- int [allocate](#) (size\_t nrows, size\_t ncols)  
*Allocate memory for size n after freeing any memory presently in use.*
- int [free](#) ()  
*Free the memory.*

### Other methods

- [omatrix\\_cx\\_tlate](#)< data\_t, parent\_t, block\_t, complex\_t > [transpose](#) ()
- [omatrix\\_cx\\_tlate](#)< data\_t, parent\_t, block\_t, complex\_t > [htranspose](#) ()  
*Compute the conjugate transpose.*

## 3.238.2 Member Function Documentation

### 3.238.2.1 int free () [inline]

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 564 of file [omatrix\\_cx\\_tlate.h](#).

### 3.238.2.2 omatrix\_cx\_tlate<data\_t,parent\_t,block\_t,complex\_t> transpose () [inline]

Compute the transpose

Definition at line 583 of file [omatrix\\_cx\\_tlate.h](#).

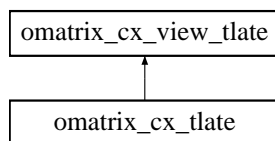
The documentation for this class was generated from the following file:

- [omatrix\\_cx\\_tlate.h](#)

## 3.239 omatrix\_cx\_view\_tlate Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for [omatrix\\_cx\\_view\\_tlate](#)::



### 3.239.1 Detailed Description

`template<class data_t, class parent_t, class block_t, class complex_t> class omatrix_cx_view_tlate< data_t, parent_t, block_t, complex_t >`

A matrix view of double-precision numbers.

Definition at line 49 of file `omatrix_cx_tlate.h`.

#### Public Member Functions

- [~omatrix\\_cx\\_view\\_tlate](#) ()

#### Copy constructors

- [omatrix\\_cx\\_view\\_tlate](#) (const [omatrix\\_cx\\_view\\_tlate](#) &v)  
*So2scflow copy constructor - create a new view of the same matrix.*
- [omatrix\\_cx\\_view\\_tlate](#) & [operator=](#) (const [omatrix\\_cx\\_view\\_tlate](#) &v)  
*So2scflow copy constructor - create a new view of the same matrix.*

#### Get and set methods

- `complex_t * operator[ ]` (size\_t i)  
*Array-like indexing.*
- `const complex_t * operator[ ]` (size\_t i) const  
*Array-like indexing.*
- `complex_t & operator()` (size\_t i, size\_t j)  
*Array-like indexing.*
- `const complex_t & operator()` (size\_t i, size\_t j) const  
*Array-like indexing.*
- `complex_t get` (size\_t i, size\_t j) const  
*Get (with optional range-checking).*
- `std::complex< data_t > get\_stl` (size\_t i, size\_t j) const  
*Get STL-like complex number (with optional range-checking).*
- `data_t real` (size\_t i, size\_t j) const  
*Get real part (with optional range-checking).*
- `data_t imag` (size\_t i, size\_t j) const  
*Get imaginary part (with optional range-checking).*
- `complex_t * get\_ptr` (size\_t i, size\_t j)  
*Get pointer (with optional range-checking).*
- `const complex_t * get\_const\_ptr` (size\_t i, size\_t j) const  
*Get pointer (with optional range-checking).*
- `int set` (size\_t i, size\_t j, complex\_t &val)  
*Set (with optional range-checking).*
- `int set` (size\_t i, size\_t j, data\_t vr, data\_t vi)  
*Set (with optional range-checking).*
- `int set\_real` (size\_t i, size\_t j, data\_t vr)  
*Set (with optional range-checking).*
- `int set\_imag` (size\_t i, size\_t j, data\_t vi)  
*Set (with optional range-checking).*
- `int set\_all` (complex\_t &val)  
*Set all.*
- `size_t rows` () const  
*Method to return number of rows.*
- `size_t cols` () const  
*Method to return number of columns.*
- `size_t tda` () const  
*Method to return matrix tda.*

#### Other methods

- `bool is\_owner` () const  
*Return true if this object owns the data it refers to.*



### 3.239.2 Member Function Documentation

#### 3.239.2.1 `size_t rows () const` [inline]

Method to return number of rows.

If no memory has been allocated, this will quietly return zero.

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

#### 3.239.2.2 `size_t cols () const` [inline]

Method to return number of columns.

If no memory has been allocated, this will quietly return zero.

Definition at line 342 of file `omatrix_cx_tlate.h`.

#### 3.239.2.3 `size_t tda () const` [inline]

Method to return matrix tda.

If no memory has been allocated, this will quietly return zero.

Definition at line 352 of file `omatrix_cx_tlate.h`.

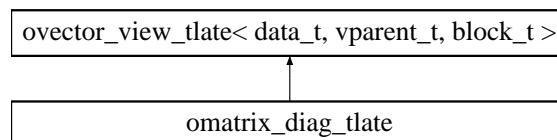
The documentation for this class was generated from the following file:

- [omatrix\\_cx\\_tlate.h](#)

## 3.240 **omatrix\_diag\_tlate** Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_diag_tlate`:



### 3.240.1 Detailed Description

```
template<class data_t, class mparent_t, class vparent_t, class block_t> class omatrix_diag_tlate< data_t, mparent_t,
vparent_t, block_t >
```

Create a vector from the main diagonal.

Definition at line 758 of file `omatrix_tlate.h`.

#### Public Member Functions

- [omatrix\\_diag\\_tlate](#) ([omatrix\\_view\\_tlate](#)< data\_t, mparent\_t, block\_t > &m)  
Create a vector of the diagonal of matrix m.

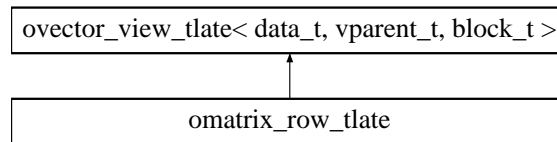
The documentation for this class was generated from the following file:

- [omatrix\\_tlate.h](#)

### 3.241 `omatrix_row_tlate` Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_row_tlate`:



#### 3.241.1 Detailed Description

```
template<class data_t, class mparent_t, class vparent_t, class block_t> class omatrix_row_tlate< data_t, mparent_t, vparent_t, block_t >
```

Create a vector from a row of a matrix.

Definition at line 669 of file `omatrix_tlate.h`.

#### Public Member Functions

- [omatrix\\_row\\_tlate](#) ([omatrix\\_view\\_tlate](#)< `data_t`, `mparent_t`, `block_t` > &`m`, `size_t` `i`)  
Create a vector from row `i` of matrix `m`.

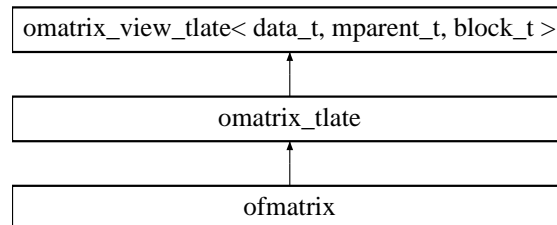
The documentation for this class was generated from the following file:

- [omatrix\\_tlate.h](#)

### 3.242 `omatrix_tlate` Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_tlate`:



#### 3.242.1 Detailed Description

```
template<class data_t, class mparent_t, class vparent_t, class block_t> class omatrix_tlate< data_t, mparent_t, vparent_t, block_t >
```

A matrix of double-precision numbers.

---

Definition at line 362 of file `omatrix_tlate.h`.

## Public Member Functions

- [~omatrix\\_tlate](#) ()

### Standard constructor

- [omatrix\\_tlate](#) (size\_t r=0, size\_t c=0)  
*Create an omatrix of size n with owner as true.*

### Copy constructors

- [omatrix\\_tlate](#) (const [omatrix\\_tlate](#) &v)  
*Deep copy constructor; allocate new space and make a copy.*
- [omatrix\\_tlate](#) (const [omatrix\\_view\\_tlate](#)< data\_t, mparent\_t, block\_t > &v)  
*Deep copy constructor; allocate new space and make a copy.*
- [omatrix\\_tlate](#) & operator= (const [omatrix\\_tlate](#) &v)  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*
- [omatrix\\_tlate](#) & operator= (const [omatrix\\_view\\_tlate](#)< data\_t, mparent\_t, block\_t > &v)  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*
- [omatrix\\_tlate](#) (size\_t n, [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > ova[ ])  
*Deep copy from an array of ovector.*
- [omatrix\\_tlate](#) (size\_t n, [uvector\\_view\\_tlate](#)< data\_t > uva[ ])  
*Deep copy from an array of uvector.*
- [omatrix\\_tlate](#) (size\_t n, size\_t n2, data\_t \*\*csa)  
*Deep copy from a C-style 2-d array.*

### Memory allocation

- int [allocate](#) (size\_t nrow, size\_t ncol)  
*Allocate memory after freeing any memory presently in use.*
- int [free](#) ()  
*Free the memory.*

### Other methods

- [omatrix\\_tlate](#)< data\_t, mparent\_t, vparent\_t, block\_t > [transpose](#) ()

## 3.242.2 Member Function Documentation

### 3.242.2.1 int free () [inline]

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 634 of file `omatrix_tlate.h`.

### 3.242.2.2 omatrix\_tlate<data\_t,mparent\_t,vparent\_t,block\_t> transpose () [inline]

Compute the transpose (even if matrix is not square)

Definition at line 653 of file `omatrix_tlate.h`.

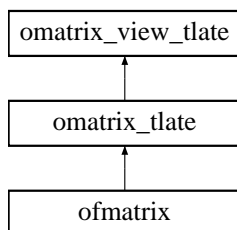
The documentation for this class was generated from the following file:

- [omatrix\\_tlate.h](#)

### 3.243 `omatrix_view_tlate` Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_view_tlate`:



#### 3.243.1 Detailed Description

```
template<class data_t, class mparent_t, class block_t> class omatrix_view_tlate< data_t, mparent_t, block_t >
```

A matrix view of double-precision numbers.

#### Todo

This class isn't sufficiently general for some applications, such as sub-matrices of higher-dimensional structures. It might be nice to create a more general class with a "stride" and a "tda".

Definition at line 54 of file `omatrix_tlate.h`.

#### Public Member Functions

- [~omatrix\\_view\\_tlate](#) ()

#### Copy constructors

- [omatrix\\_view\\_tlate](#) (const [omatrix\\_view\\_tlate](#) &v)  
*So2scflow copy constructor - create a new view of the same matrix.*
- [omatrix\\_view\\_tlate](#) & [operator=](#) (const [omatrix\\_view\\_tlate](#) &v)  
*So2scflow copy constructor - create a new view of the same matrix.*

#### Get and set methods

- `data_t * operator[ ]` (size\_t i)  
*Array-like indexing.*
- `const data_t * operator[ ]` (size\_t i) const  
*Array-like indexing.*
- `data_t & operator()` (size\_t i, size\_t j)  
*Array-like indexing.*
- `const data_t & operator()` (size\_t i, size\_t j) const  
*Array-like indexing.*
- `data_t get` (size\_t i, size\_t j) const  
*Get (with optional range-checking).*
- `data_t * get\_ptr` (size\_t i, size\_t j)  
*Get pointer (with optional range-checking).*
- `const data_t * get\_const\_ptr` (size\_t i, size\_t j) const  
*Get pointer (with optional range-checking).*
- `int set` (size\_t i, size\_t j, data\_t val)

*Set (with optional range-checking).*

- int [set\\_all](#) (double val)  
*Set all of the value to be the value val.*
- size\_t [rows](#) () const  
*Method to return number of rows.*
- size\_t [cols](#) () const  
*Method to return number of columns.*
- size\_t [tda](#) () const  
*Method to return matrix tda.*

### Other methods

- bool [is\\_owner](#) () const  
*Return true if this object owns the data it refers to.*
- mparent\_t \* [get\\_gsl\\_matrix](#) ()  
*Return a gsl matrix.*
- const mparent\_t \* [get\\_gsl\\_matrix\\_const](#) () const  
*Return a const gsl matrix.*

### Arithmetic

- [omatrix\\_view\\_tlate](#)< data\_t, mparent\_t, block\_t > & [operator+=](#) (const [omatrix\\_view\\_tlate](#)< data\_t, mparent\_t, block\_t > &x)  
*operator+=*
- [omatrix\\_view\\_tlate](#)< data\_t, mparent\_t, block\_t > & [operator-=](#) (const [omatrix\\_view\\_tlate](#)< data\_t, mparent\_t, block\_t > &x)  
*operator-=*
- [omatrix\\_view\\_tlate](#)< data\_t, mparent\_t, block\_t > & [operator+=](#) (const data\_t &y)  
*operator+=*
- [omatrix\\_view\\_tlate](#)< data\_t, mparent\_t, block\_t > & [operator-=](#) (const data\_t &y)  
*operator-=*
- [omatrix\\_view\\_tlate](#)< data\_t, mparent\_t, block\_t > & [operator\\*=](#) (const data\_t &y)  
*operator\*=*

### Protected Member Functions

- [omatrix\\_view\\_tlate](#) ()  
*Empty constructor provided for use by [omatrix\\_tlate](#)(const [omatrix\\_tlate](#) &v).*

## 3.243.2 Member Function Documentation

### 3.243.2.1 size\_t rows () const [inline]

Method to return number of rows.

If no memory has been allocated, this will quietly return zero.

Definition at line 236 of file [omatrix\\_tlate.h](#).

### 3.243.2.2 size\_t cols () const [inline]

Method to return number of columns.

If no memory has been allocated, this will quietly return zero.

Definition at line 246 of file [omatrix\\_tlate.h](#).

**3.243.2.3** `size_t tda () const` `[inline]`

Method to return matrix tda.

If no memory has been allocated, this will quietly return zero.

Definition at line 256 of file `omatrix_tlate.h`.

The documentation for this class was generated from the following file:

- [omatrix\\_tlate.h](#)

**3.244 other\_ioc Class Reference**

```
#include <other_ioc.h>
```

**3.244.1 Detailed Description**

Setup I/O for series acceleration.

Definition at line 36 of file `other_ioc.h`.

**Public Member Functions**

- [other\\_ioc \(\)](#)
- [~other\\_ioc \(\)](#)

**Data Fields**

- [gsl\\_series\\_io\\_type \\* gsl\\_series\\_io](#)

The documentation for this class was generated from the following file:

- `other_ioc.h`

**3.245 other\_todos\_and\_bugs Class Reference**

```
#include <main.h>
```

**3.245.1 Detailed Description**

An empty class to add some items to the todo and bug lists.

**Todo**

- Fix problems with `-ansi` compilation on Cygwin
  - More examples and benchmarks
  - There is a problem with const-correctness in vectors. I'm not sure how it's best solved. It could be best to create two kinds of `ovector_view`'s: one const and one not.
  - Document a list of all global functions and operators
  - Make sure we have a `uvector_alloc`, `uvector_cx_alloc`, `ovector_cx_const_reverse`, `ovector_cx_const_subvector_reverse`, `uvector_reverse`, `uvector_const_reverse`, `uvector_subvector_reverse`, `uvector_const_subvector_reverse`, `omatrix_cx_diag`, `blah_const_diag`, `umatrix_diag`, and `umatrix_cx_diag`
-

- `ovector_cx_view::operator=(uvector_cx_view &)` is missing
- `ovector_cx::operator=(uvector_cx_view &)` is missing
- `uvector_c_view::operator+=(complex)` is missing
- `uvector_c_view::operator-=(complex)` is missing
- `uvector_c_view::operator*=(complex)` is missing

### Bug

- The file `configure.ac` does not correctly produce an error if the argument `--disable-readline` is not given when the `libncurses` and `libreadline` libraries are not present.
- BLAS libraries not named `libblas` or `libgslibblas` are not properly detected in `./configure` and will have to be added manually.
- The `-lm` flag may not be added properly by `./configure`

Definition at line 1554 of file `main.h`.

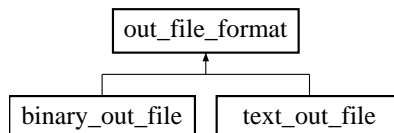
The documentation for this class was generated from the following file:

- `main.h`

## 3.246 out\_file\_format Class Reference

```
#include <file_format.h>
```

Inheritance diagram for `out_file_format`:



### 3.246.1 Detailed Description

Abstract base class for output file formats.

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

#### Public Member Functions

- virtual `~out_file_format()`
- virtual int `bool_out` (bool dat, std::string name="")=0  
*Output a bool variable.*
- virtual int `char_out` (char dat, std::string name="")=0  
*Output a char variable.*
- virtual int `float_out` (float dat, std::string name="")=0  
*Output a float variable.*
- virtual int `double_out` (double dat, std::string name="")=0  
*Output a double variable.*
- virtual int `int_out` (int dat, std::string name="")=0  
*Output an int variable.*
- virtual int `long_out` (unsigned long int dat, std::string name="")=0  
*Output an long variable.*

- virtual int [string\\_out](#) (std::string dat, std::string name="")=0  
*Output a string.*
- virtual int [word\\_out](#) (std::string dat, std::string name="")=0  
*Output a word.*
- virtual int [start\\_object](#) (std::string type, std::string name="")=0  
*Start object output.*
- virtual int [end\\_object](#) ()=0  
*End object output.*
- virtual int [end\\_line](#) ()=0  
*End a line of output.*
- virtual int [init\\_file](#) ()=0  
*Output initialization.*
- virtual int [clean\\_up](#) ()=0  
*Finish the file.*

The documentation for this class was generated from the following file:

- [file\\_format.h](#)

## 3.247 **ovector\_alloc** Class Reference

```
#include <ovector_tlate.h>
```

### 3.247.1 Detailed Description

A simple class to provide an [allocate\(\)](#) function for [ovector](#).

Definition at line 1849 of file [ovector\\_tlate.h](#).

#### Public Member Functions

- void [allocate](#) ([ovector](#) &o, int i)  
*Allocate  $v$  for  $i$  elements.*
- void [free](#) ([ovector](#) &o)  
*Free memory.*

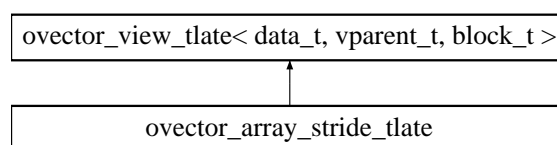
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.248 **ovector\_array\_stride\_tlate** Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for [ovector\\_array\\_stride\\_tlate](#):





### 3.248.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t> class ovector\_array\_stride\_tlate< data\_t, vparent\_t, block\_t >**

Create a vector from an array with a stride.

Definition at line 1017 of file ovector\_tlate.h.

#### Public Member Functions

- [ovector\\_array\\_stride\\_tlate](#) (size\_t n, data\_t \*dat, size\_t s)  
*Create a vector from dat with size n and stride s.*

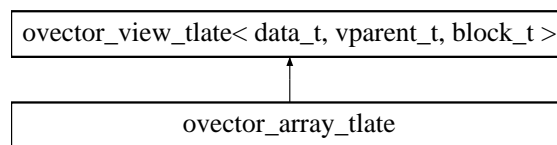
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.249 ovector\_array\_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector\_array\_tlate::



### 3.249.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t> class ovector\_array\_tlate< data\_t, vparent\_t, block\_t >**

Create a vector from an array.

Definition at line 996 of file ovector\_tlate.h.

#### Public Member Functions

- [ovector\\_array\\_tlate](#) (size\_t n, data\_t \*dat)  
*Create a vector from dat with size n.*

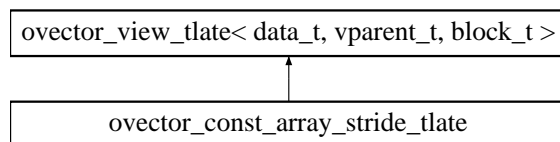
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.250 ovector\_const\_array\_stride\_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector\_const\_array\_stride\_tlate::



### 3.250.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t> class ovector\_const\_array\_stride\_tlate< data\_t, vparent\_t, block\_t >**

Create a const vector from an array with a stride.

Definition at line 1123 of file ovector\_tlate.h.

#### Public Member Functions

- [ovector\\_const\\_array\\_stride\\_tlate](#) (size\_t n, const data\_t \*dat, size\_t s)  
*Create a vector from dat with size n.*

#### Protected Member Functions

Ensure \c const by hiding non-const members

- data\_t & [operator\[\]](#) (size\_t i)  
*Array-like indexing.*
- data\_t & [operator\(\)](#) (size\_t i)  
*Array-like indexing.*
- data\_t \* [get\\_ptr](#) (size\_t i)  
*Get pointer (with optional range-checking).*
- int [set](#) (size\_t i, data\_t val)  
*Set (with optional range-checking).*
- int [swap](#) ([ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*Swap vectors.*
- int [set\\_all](#) (double val)  
*Set all of the value to be the value val.*
- vparent\_t \* [get\\_gsl\\_vector](#) ()  
*Return a gsl vector.*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator+=](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*operator+=*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator-=](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*operator-=*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator \\*=](#) (const data\_t &y)  
*operator\*=*

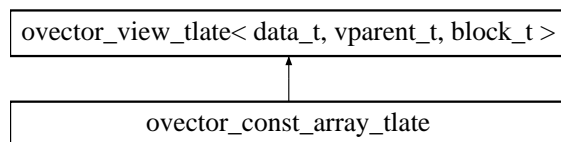
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.251 ovector\_const\_array\_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector\_const\_array\_tlate::



### 3.251.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t> class ovector\_const\_array\_tlate< data\_t, vparent\_t, block\_t >**

Create a const vector from an array.

Definition at line 1066 of file ovector\_tlate.h.

#### Public Member Functions

- [ovector\\_const\\_array\\_tlate](#) (size\_t n, const data\_t \*dat)  
*Create a vector from dat with size n.*
- [~ovector\\_const\\_array\\_tlate](#) ()

#### Protected Member Functions

##### Ensure \c const by hiding non-const members

- data\_t & [operator\[\]](#) (size\_t i)  
*Array-like indexing.*
- data\_t & [operator\(\)](#) (size\_t i)  
*Array-like indexing.*
- data\_t \* [get\\_ptr](#) (size\_t i)  
*Get pointer (with optional range-checking).*
- int [set](#) (size\_t i, data\_t val)  
*Set (with optional range-checking).*
- int [swap](#) ([ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*Swap vectors.*
- int [set\\_all](#) (double val)  
*Set all of the value to be the value val.*
- vparent\_t \* [get\\_gsl\\_vector](#) ()  
*Return a gsl vector.*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator+=](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*operator+=*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator-=](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*operator-=*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator\\*==](#) (const data\_t &y)  
*operator\*==*

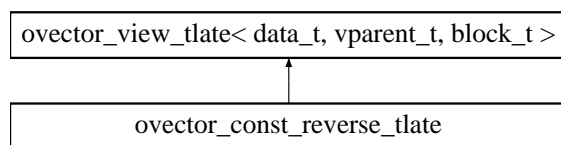
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.252 ovector\_const\_reverse\_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ovector_const_reverse_tlate`::



### 3.252.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t> class `ovector_const_reverse_tlate`< data\_t, vparent\_t, block\_t >**

Reversed view of a vector.

Definition at line 1411 of file `ovector_tlate.h`.

#### Public Member Functions

- [`ovector\_const\_reverse\_tlate`](#) (const [`ovector\_view\_tlate`](#)< data\_t, vparent\_t, block\_t > &v)  
*Create a vector from dat with size n and stride s.*

#### Get and set methods

- const data\_t & [`operator\[\]`](#) (size\_t i) const  
*Array-like indexing.*
- const data\_t & [`operator\(\)`](#) (size\_t i) const  
*Array-like indexing.*
- data\_t [`get`](#) (size\_t i) const  
*Get (with optional range-checking).*
- const data\_t \* [`get\_const\_ptr`](#) (size\_t i) const  
*Get pointer (with optional range-checking).*

### 3.252.2 Member Function Documentation

#### 3.252.2.1 `const data_t& operator[] (size_t i) const` [inline]

Array-like indexing.

Array-like indexing

Reimplemented from [`ovector\_view\_tlate`](#).

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

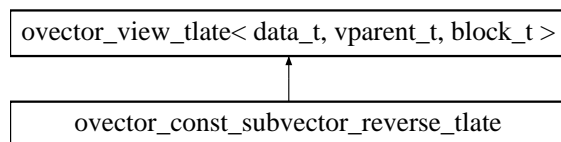
The documentation for this class was generated from the following file:

- [`ovector\_tlate.h`](#)

## 3.253 `ovector_const_subvector_reverse_tlate` Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ovector_const_subvector_reverse_tlate`::



### 3.253.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t> class ovector\_const\_subvector\_reverse\_tlate< data\_t, vparent\_t, block\_t >**

Reversed view of a const subvector.

Definition at line 1663 of file `ovector_tlate.h`.

#### Public Member Functions

- [ovector\\_const\\_subvector\\_reverse\\_tlate](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &v, size\_t offset, size\_t n)  
*Create a vector from dat with size n and stride s.*

#### Get and set methods

- const data\_t & [operator\[\]](#) (size\_t i) const  
*Array-like indexing.*
- const data\_t & [operator\(\)](#) (size\_t i) const  
*Array-like indexing.*
- data\_t [get](#) (size\_t i) const  
*Get (with optional range-checking).*
- const data\_t \* [get\\_const\\_ptr](#) (size\_t i) const  
*Get pointer (with optional range-checking).*

### 3.253.2 Member Function Documentation

#### 3.253.2.1 const data\_t& operator[] (size\_t i) const [inline]

Array-like indexing.

Array-like indexing

Reimplemented from [ovector\\_view\\_tlate](#).

Definition at line 1692 of file `ovector_tlate.h`.

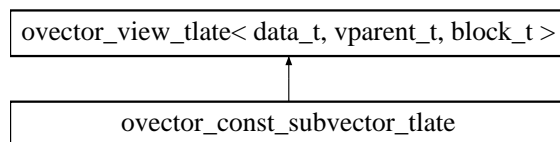
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.254 `ovector_const_subvector_tlate` Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ovector_const_subvector_tlate`:



### 3.254.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t> class ovector\_const\_subvector\_tlate< data\_t, vparent\_t, block\_t >**

Create a const vector from a subvector of another vector.

Definition at line 1178 of file ovector\_tlate.h.

#### Public Member Functions

- [ovector\\_const\\_subvector\\_tlate](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &orig, size\_t offset, size\_t n)  
*Create a vector from orig.*
- const data\_t & [operator\[\]](#) (size\_t i) const  
*Array-like indexing.*

#### Protected Member Functions

##### Ensure \c const by hiding non-const members

- data\_t & [operator\[\]](#) (size\_t i)  
*Array-like indexing.*
- data\_t & [operator\(\)](#) (size\_t i)  
*Array-like indexing.*
- data\_t \* [get\\_ptr](#) (size\_t i)  
*Get pointer (with optional range-checking).*
- int [set](#) (size\_t i, data\_t val)  
*Set (with optional range-checking).*
- int [swap](#) ([ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*Swap vectors.*
- int [set\\_all](#) (double val)  
*Set all of the value to be the value val.*
- vparent\_t \* [get\\_gsl\\_vector](#) ()  
*Return a gsl vector.*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator+=](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*operator+=*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator-=](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*operator-=*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator \\*=](#) (const data\_t &y)  
*operator\*=*

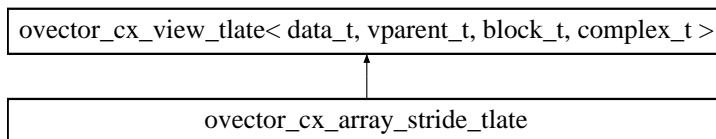
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

### 3.255 ovector\_cx\_array\_stride\_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector\_cx\_array\_stride\_tlate::



#### 3.255.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t, class complex\_t> class ovector\_cx\_array\_stride\_tlate< data\_t, vparent\_t, block\_t, complex\_t >**

Create a vector from an array with a stride.

Definition at line 742 of file ovector\_cx\_tlate.h.

#### Public Member Functions

- [ovector\\_cx\\_array\\_stride\\_tlate](#) (size\_t n, complex\_t \*dat, size\_t s)  
*Create a vector from dat with size n and stride s.*

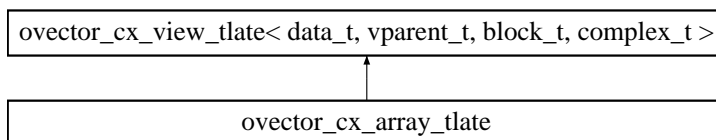
The documentation for this class was generated from the following file:

- [ovector\\_cx\\_tlate.h](#)

### 3.256 ovector\_cx\_array\_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector\_cx\_array\_tlate::



#### 3.256.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t, class complex\_t> class ovector\_cx\_array\_tlate< data\_t, vparent\_t, block\_t, complex\_t >**

Create a vector from an array.

Definition at line 724 of file ovector\_cx\_tlate.h.

## Public Member Functions

- [`ovector\_cx\_array\_tlate`](#) (`size_t n`, `complex_t *dat`)  
*Create a vector from `dat` with size `n`.*

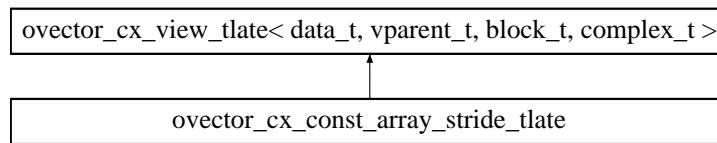
The documentation for this class was generated from the following file:

- [ovector\\_cx\\_tlate.h](#)

## 3.257 `ovector_cx_const_array_stride_tlate` Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_const_array_stride_tlate`:



### 3.257.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t, class complex\_t> class `ovector_cx_const_array_stride_tlate`< data\_t, vparent\_t, block\_t, complex\_t >**

Create a vector from an `array_stride`.

Definition at line 843 of file `ovector_cx_tlate.h`.

## Public Member Functions

- [`ovector\_cx\_const\_array\_stride\_tlate`](#) (`size_t n`, `const complex_t *dat`, `size_t s`)  
*Create a vector from `dat` with size `n`.*

## Protected Member Functions

These are inaccessible to ensure the vector is `\c const`.

- `data_t & operator[]` (`size_t i`)  
*Array-like indexing.*
- `data_t & operator()` (`size_t i`)  
*Array-like indexing.*
- `data_t * get_ptr` (`size_t i`)  
*Get pointer (with optional range-checking).*
- `int set` (`size_t i`, `data_t val`)
- `int swap` (`ovector_cx_view_tlate`< `data_t`, `vparent_t`, `block_t`, `complex_t` > &`x`)
- `int set_all` (`double val`)
- `vparent_t * get_gsl_vector` ()
- `ovector_cx_view_tlate`< `data_t`, `vparent_t`, `block_t`, `complex_t` > & `operator+=` (`const ovector_cx_view_tlate`< `data_t`, `vparent_t`, `block_t`, `complex_t` > &`x`)  
*operator+=*
- `ovector_cx_view_tlate`< `data_t`, `vparent_t`, `block_t`, `complex_t` > & `operator-=` (`const ovector_cx_view_tlate`< `data_t`, `vparent_t`, `block_t`, `complex_t` > &`x`)  
*operator-=*



- `ovector_cx_view_tlate`< data\_t, vparent\_t, block\_t, complex\_t > & `operator *=` (const data\_t &y)  
*operator\*=*

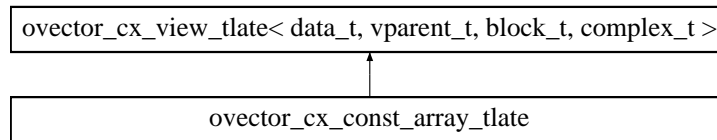
The documentation for this class was generated from the following file:

- [ovector\\_cx\\_tlate.h](#)

## 3.258 `ovector_cx_const_array_tlate` Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_const_array_tlate`:



### 3.258.1 Detailed Description

```
template<class data_t, class vparent_t, class block_t, class complex_t> class ovector_cx_const_array_tlate< data_t, vparent_t, block_t, complex_t >
```

Create a vector from an array.

Definition at line 788 of file `ovector_cx_tlate.h`.

#### Public Member Functions

- `ovector_cx_const_array_tlate` (size\_t n, const complex\_t \*dat)  
*Create a vector from dat with size n.*
- `~ovector_cx_const_array_tlate` ()

#### Protected Member Functions

These are inaccessible to ensure the vector is `\c const`.

- data\_t & `operator[]` (size\_t i)  
*Array-like indexing.*
- data\_t & `operator()` (size\_t i)  
*Array-like indexing.*
- data\_t \* `get_ptr` (size\_t i)  
*Get pointer (with optional range-checking).*
- int `set` (size\_t i, data\_t val)
- int `swap` (`ovector_cx_view_tlate`< data\_t, vparent\_t, block\_t, complex\_t > &x)
- int `set_all` (double val)
- vparent\_t \* `get_gsl_vector` ()
- `ovector_cx_view_tlate`< data\_t, vparent\_t, block\_t, complex\_t > & `operator+=` (const `ovector_cx_view_tlate`< data\_t, vparent\_t, block\_t, complex\_t > &x)  
*operator+=*
- `ovector_cx_view_tlate`< data\_t, vparent\_t, block\_t, complex\_t > & `operator-=` (const `ovector_cx_view_tlate`< data\_t, vparent\_t, block\_t, complex\_t > &x)  
*operator-=*
- `ovector_cx_view_tlate`< data\_t, vparent\_t, block\_t, complex\_t > & `operator *=` (const data\_t &y)  
*operator\*=*

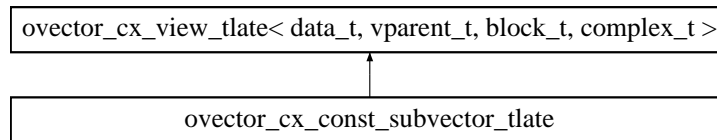
The documentation for this class was generated from the following file:

- [ovector\\_cx\\_tlate.h](#)

### 3.259 `ovector_cx_const_subvector_tlate` Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_const_subvector_tlate`:



#### 3.259.1 Detailed Description

```
template<class data_t, class vparent_t, class block_t, class complex_t> class ovector_cx_const_subvector_tlate< data_t,
vparent_t, block_t, complex_t >
```

Create a vector from a subvector of another.

Definition at line 897 of file `ovector_cx_tlate.h`.

#### Public Member Functions

- [ovector\\_cx\\_const\\_subvector\\_tlate](#) (const [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > &orig, size\_t offset, size\_t n)  
*Create a vector from orig.*

#### Protected Member Functions

These are inaccessible to ensure the vector is `\c const`.

- data\_t & [operator\[\]](#) (size\_t i)  
*Array-like indexing.*
- data\_t & [operator\(\)](#) (size\_t i)  
*Array-like indexing.*
- data\_t \* [get\\_ptr](#) (size\_t i)  
*Get pointer (with optional range-checking).*
- int [set](#) (size\_t i, data\_t val)
- int [swap](#) ([ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > &x)
- int [set\\_all](#) (double val)
- vparent\_t \* [get\\_gsl\\_vector](#) ()
- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator+=](#) (const [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > &x)  
*operator+=*
- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator-=](#) (const [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > &x)  
*operator-=*
- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator\\*==](#) (const data\_t &y)  
*operator\*==*

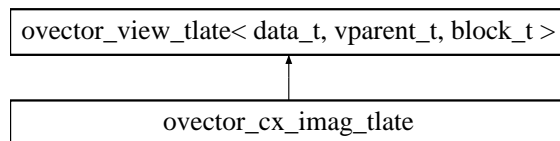
The documentation for this class was generated from the following file:

- [ovector\\_cx\\_tlate.h](#)

## 3.260 `ovector_cx_imag_tlate` Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_imag_tlate`:



### 3.260.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t, class cvparent\_t, class cblock\_t, class complex\_t> class `ovector_cx_imag_tlate`< data\_t, vparent\_t, block\_t, cvparent\_t, cblock\_t, complex\_t >**

Create a imaginary vector from the imaginary parts of a complex vector.

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

#### Public Member Functions

- [`ovector\_cx\_imag\_tlate`](#) ([`ovector\_cx\_view\_tlate`](#)< data\_t, cvparent\_t, cblock\_t, complex\_t > &x)  
*Create a imaginary vector from the imaginary parts of a complex vector.*

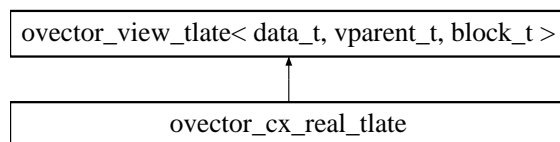
The documentation for this class was generated from the following file:

- [ovector\\_cx\\_tlate.h](#)

## 3.261 `ovector_cx_real_tlate` Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_real_tlate`:



### 3.261.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t, class cvparent\_t, class cblock\_t, class complex\_t> class `ovector_cx_real_tlate`< data\_t, vparent\_t, block\_t, cvparent\_t, cblock\_t, complex\_t >**

Create a real vector from the real parts of a complex vector.

Definition at line 955 of file `ovector_cx_tlate.h`.

## Public Member Functions

- [`ovector\_cx\_real\_tlate`](#) ([`ovector\_cx\_view\_tlate`](#)< `data_t`, `cvparent_t`, `cblock_t`, `complex_t` > &`x`)  
*Create a real vector from the real parts of a complex vector.*

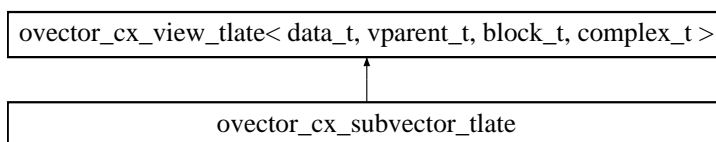
The documentation for this class was generated from the following file:

- [`ovector\_cx\_tlate.h`](#)

## 3.262 `ovector_cx_subvector_tlate` Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_subvector_tlate`:



### 3.262.1 Detailed Description

**template<class `data_t`, class `vparent_t`, class `block_t`, class `complex_t`> class `ovector_cx_subvector_tlate`< `data_t`, `vparent_t`, `block_t`, `complex_t` >**

Create a vector from a subvector of another.

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

## Public Member Functions

- [`ovector\_cx\_subvector\_tlate`](#) ([`ovector\_cx\_view\_tlate`](#)< `data_t`, `vparent_t`, `block_t`, `complex_t` > &`orig`, `size_t` `offset`, `size_t` `n`)  
*Create a vector from `orig`.*

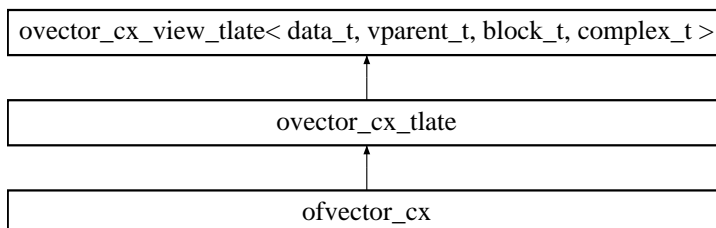
The documentation for this class was generated from the following file:

- [`ovector\_cx\_tlate.h`](#)

## 3.263 `ovector_cx_tlate` Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_tlate`:



### 3.263.1 Detailed Description

`template<class data_t, class vparent_t, class block_t, class complex_t> class ovector_cx_tlate< data_t, vparent_t, block_t, complex_t >`

A vector of double-precision numbers.

If the memory allocation fails, either in the constructor or in `allocate()`, then the error handler will be called, partially allocated memory will be freed, and the size will be reset to zero. You can test to see if the allocation succeeded using something like

```
const size_t n=10;
ovector_cx x(10);
if (x.size()==0) cout << "Failed." << endl;
```

#### Todo

There is a slight difference between how this works in comparison to MV++. The function `allocate()` operates a little differently than `newsize()`, as it will feel free to allocate new memory when owner is false. It's not clear if this is an issue, however, since it doesn't appear possible to create an `ovector_cx_tlate` with a value of `owner` equal to zero. This situation ought to be clarified further.

#### Todo

Add `subvector_stride`, `const_subvector_stride`

Definition at line 490 of file `ovector_cx_tlate.h`.

### Public Member Functions

- `~ovector_cx_tlate ()`

#### Standard constructor

- `ovector_cx_tlate (size_t n=0)`  
*Create an `ovector_cx` of size `n` with owner as 'true'.*

#### Copy constructors

- `ovector_cx_tlate (const ovector_cx_tlate &v)`  
*Deep copy constructor; allocate new space and make a copy.*
- `ovector_cx_tlate (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &v)`  
*Deep copy constructor; allocate new space and make a copy.*
- `ovector_cx_tlate & operator= (const ovector_cx_tlate &v)`  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*
- `ovector_cx_tlate & operator= (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &v)`  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*

#### Memory allocation

- `int allocate (size_t nsize)`  
*Allocate memory for size `n` after freeing any memory presently in use.*
- `int free ()`  
*Free the memory.*

#### Other methods

- `vparent_t * get_gsl_vector_complex ()`  
*Return a `gsl` vector\_cx.*
- `const vparent_t * get_gsl_vector_complex_const () const`  
*Return a `gsl` vector\_cx.*

### 3.263.2 Member Function Documentation

#### 3.263.2.1 int free() [inline]

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 694 of file ovector\_cx\_tlate.h.

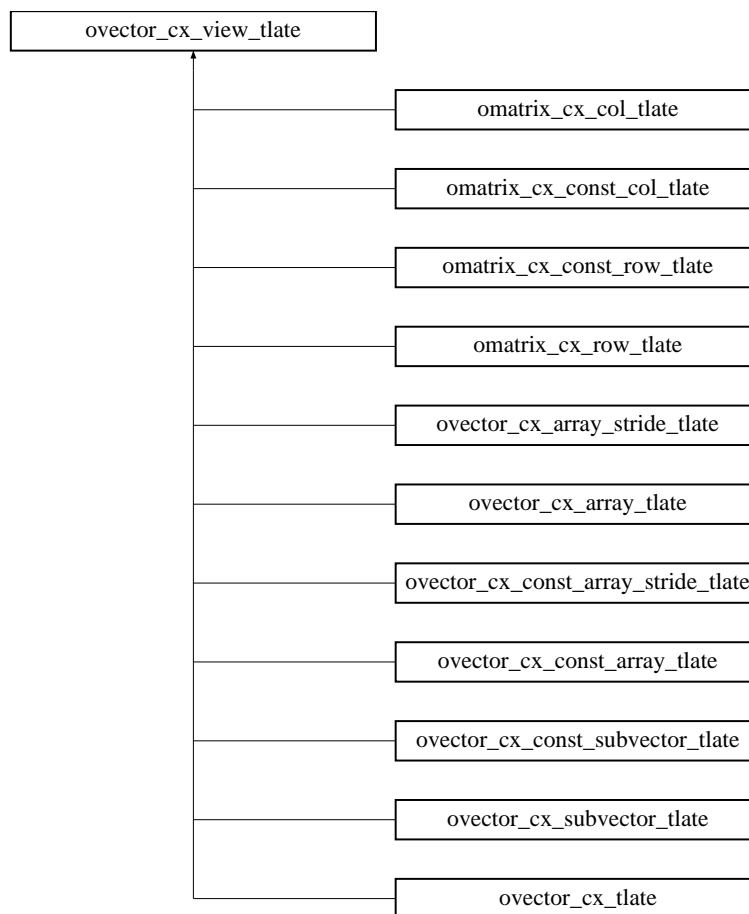
The documentation for this class was generated from the following file:

- [ovector\\_cx\\_tlate.h](#)

## 3.264 ovector\_cx\_view\_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector\_cx\_view\_tlate::



### 3.264.1 Detailed Description

```
template<class data_t, class vparent_t, class block_t, class complex_t> class ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t >
```

A vector view of double-precision numbers.

## Todo

Move conversion b/w `complex<double>` and `gsl_complex` to [cx\\_arith.h](#)

Definition at line 59 of file `ovector_cx_tlate.h`.

## Public Member Functions

- `~ovector_cx_view_tlate` ()
- `int conjugate` ()  
*Conjugate the vector.*
- `data_t norm` () const  
*Complex norm  $v^\dagger v$ .*

## Copy constructors

- `ovector_cx_view_tlate` (const `ovector_cx_view_tlate` &*v*)  
*So2scllow copy constructor - create a new view of the same vector.*
- `ovector_cx_view_tlate & operator=` (const `ovector_cx_view_tlate` &*v*)  
*So2scllow copy constructor - create a new view of the same vector.*

## Get and set methods

- `complex_t & operator[]` (size\_t *i*)  
*Array-like indexing.*
- `const complex_t & operator[]` (size\_t *i*) const  
*Array-like indexing.*
- `complex_t & operator()` (size\_t *i*)  
*Array-like indexing.*
- `const complex_t & operator()` (size\_t *i*) const  
*Array-like indexing.*
- `complex_t get` (size\_t *i*) const  
*Get (with optional range-checking).*
- `data_t real` (size\_t *i*) const  
*Get real part (with optional range-checking).*
- `data_t imag` (size\_t *i*) const  
*Get imaginary part (with optional range-checking).*
- `std::complex< data_t > get_stl` (size\_t *i*) const  
*Get STL-like complex number (with optional range-checking).*
- `complex_t * get_ptr` (size\_t *i*)  
*Get pointer (with optional range-checking).*
- `const complex_t * get_const_ptr` (size\_t *i*) const  
*Get pointer (with optional range-checking).*
- `int set` (size\_t *i*, const `complex_t` &*val*)  
*Set (with optional range-checking).*
- `int set_stl` (size\_t *i*, const `std::complex< data_t >` &*d*)  
*Set (with optional range-checking).*
- `int set` (size\_t *i*, `data_t` *vr*, `data_t` *vi*)  
*Set (with optional range-checking).*
- `int set_all` (const `complex_t` &*g*)  
*Set all of the value to be the value val.*
- `size_t size` () const  
*Method to return vector size.*
- `size_t stride` () const  
*Method to return vector stride.*

## Other methods

- `bool is_owner` () const  
*Return true if this object owns the data it refers to.*

## Arithmetic

- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator+=](#) (const [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > &x)  
*operator+=*
- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator-=](#) (const [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > &x)  
*operator-=*
- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator+=](#) (const complex\_t &x)  
*operator+=*
- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator-=](#) (const complex\_t &x)  
*operator-=*
- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator \\*=](#) (const complex\_t &x)  
*operator\*=*
- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator+=](#) (const data\_t &x)  
*operator+=*
- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator-=](#) (const data\_t &x)  
*operator-=*
- [ovector\\_cx\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t, complex\_t > & [operator \\*=](#) (const data\_t &x)  
*operator\*=*

## Protected Member Functions

- [ovector\\_cx\\_view\\_tlate](#) ()  
*Empty constructor provided for use by ovector\_cx\_tlate(const ovector\_cx\_tlate &v) [protected].*

### 3.264.2 Member Function Documentation

#### 3.264.2.1 size\_t size () const [inline]

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 303 of file ovector\_cx\_tlate.h.

#### 3.264.2.2 size\_t stride () const [inline]

Method to return vector stride.

If no memory has been allocated, this will quietly return zero.

Definition at line 313 of file ovector\_cx\_tlate.h.

The documentation for this class was generated from the following file:

- [ovector\\_cx\\_tlate.h](#)

## 3.265 ovector\_int\_alloc Class Reference

```
#include <ovector_tlate.h>
```

### 3.265.1 Detailed Description

A simple class to provide an [allocate\(\)](#) function for [ovector\\_int](#).

Definition at line 1860 of file ovector\_tlate.h.



## Public Member Functions

- void [allocate](#) ([ovector\\_int](#) &o, int i)  
*Allocate  $\forall$  for  $i$  elements.*
- void [free](#) ([ovector\\_int](#) &o)  
*Free memory.*

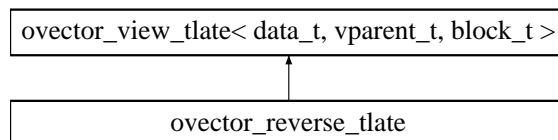
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.266 ovector\_reverse\_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector\_reverse\_tlate::



### 3.266.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t> class ovector\_reverse\_tlate< data\_t, vparent\_t, block\_t >**

Reversed view of a vector.

Note that you cannot reverse a reversed vector and expect to get the original vector back.

Definition at line 1259 of file ovector\_tlate.h.

## Public Member Functions

- [ovector\\_reverse\\_tlate](#) ([ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &v)  
*Create a vector from dat with size n and stride s.*

## Get and set methods

- data\_t & [operator\[\]](#) (size\_t i)  
*Array-like indexing.*
- const data\_t & [operator\[\]](#) (size\_t i) const  
*Array-like indexing.*
- data\_t & [operator\(\)](#) (size\_t i)  
*Array-like indexing.*
- const data\_t & [operator\(\)](#) (size\_t i) const  
*Array-like indexing.*
- data\_t [get](#) (size\_t i) const  
*Get (with optional range-checking).*
- data\_t \* [get\\_ptr](#) (size\_t i)  
*Get pointer (with optional range-checking).*
- const data\_t \* [get\\_const\\_ptr](#) (size\_t i) const  
*Get pointer (with optional range-checking).*
- int [set](#) (size\_t i, data\_t val)

- *Set (with optional range-checking).*  
 • `int set_all` (double val)  
*Set all of the value to be the value val.*

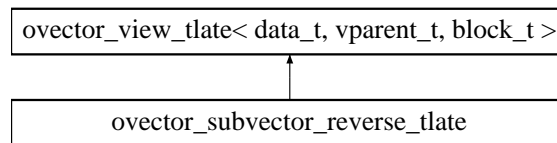
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.267 `ovector_subvector_reverse_tlate` Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ovector_subvector_reverse_tlate`:



### 3.267.1 Detailed Description

**template<class data\_t, class vparent\_t, class block\_t> class `ovector_subvector_reverse_tlate`< data\_t, vparent\_t, block\_t >**

Reversed view of a subvector.

Note that you cannot reverse a reversed vector and expect to get the original vector back.

Definition at line 1502 of file `ovector_tlate.h`.

### Public Member Functions

- [ovector\\_subvector\\_reverse\\_tlate](#) ([ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &v, size\_t offset, size\_t n)  
*Create a vector from dat with size n and stride s.*

### Get and set methods

- data\_t & [operator\[\]](#) (size\_t i)  
*Array-like indexing.*
- const data\_t & [operator\[\]](#) (size\_t i) const  
*Array-like indexing.*
- data\_t & [operator\(\)](#) (size\_t i)  
*Array-like indexing.*
- const data\_t & [operator\(\)](#) (size\_t i) const  
*Array-like indexing.*
- data\_t [get](#) (size\_t i) const  
*Get (with optional range-checking).*
- data\_t \* [get\\_ptr](#) (size\_t i)  
*Get pointer (with optional range-checking).*
- const data\_t \* [get\\_const\\_ptr](#) (size\_t i) const  
*Get pointer (with optional range-checking).*
- int [set](#) (size\_t i, data\_t val)  
*Set (with optional range-checking).*
- int [set\\_all](#) (double val)  
*Set all of the value to be the value val.*

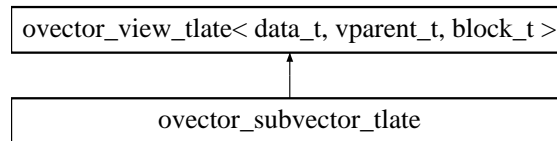
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.268 ovector\_subvector\_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector\_subvector\_tlate::



### 3.268.1 Detailed Description

```
template<class data_t, class vparent_t, class block_t> class ovector_subvector_tlate< data_t, vparent_t, block_t >
```

Create a vector from a subvector of another.

Definition at line 1039 of file ovector\_tlate.h.

#### Public Member Functions

- [ovector\\_subvector\\_tlate](#) ([ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &orig, size\_t offset, size\_t n)  
*Create a vector from orig.*

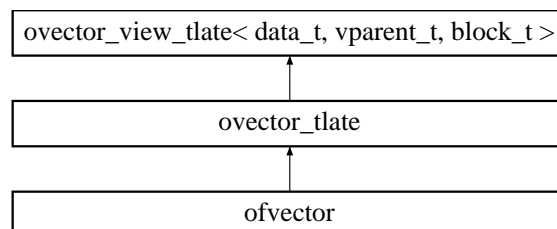
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

## 3.269 ovector\_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector\_tlate::



### 3.269.1 Detailed Description

```
template<class data_t, class vparent_t, class block_t> class ovector_tlate< data_t, vparent_t, block_t >
```

A vector with finite stride.

There are several global binary operators associated with objects of type [ovector\\_tlate](#). The are documented in the "Functions" section of [ovector\\_tlate.h](#).

Definition at line 529 of file ovector\_tlate.h.

## Public Member Functions

- [~ovector\\_tlate](#) ()

### Standard constructor

- [ovector\\_tlate](#) (size\_t n=0)  
*Create an ovector of size n with owner as 'true'.*

### Copy constructors

- [ovector\\_tlate](#) (const [ovector\\_tlate](#) &v)  
*Deep copy constructor; allocate new space and make a copy.*
- [ovector\\_tlate](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &v)  
*Deep copy constructor; allocate new space and make a copy.*
- [ovector\\_tlate](#) (const [uvector\\_view\\_tlate](#)< data\_t > &v)  
*Deep copy constructor; allocate new space and make a copy.*
- [ovector\\_tlate](#) & [operator=](#) (const [ovector\\_tlate](#) &v)  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*
- [ovector\\_tlate](#) & [operator=](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &v)  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*
- [ovector\\_tlate](#) & [operator=](#) (const [uvector\\_view\\_tlate](#)< data\_t > &v)  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*

### Memory allocation

- int [allocate](#) (size\_t nsize)  
*Allocate memory for size n after freeing any memory presently in use.*
- int [free](#) ()  
*Free the memory.*

### Stack-like operations (very experimental)

- int [push\\_back](#) (data\_t val)  
*Add a value to the end of the vector.*
- int [reserve](#) (size\_t cap)  
*Reserve memory by increasing capacity.*
- data\_t [pop](#) ()  
*Return the last value and shrink the vector size by one.*

### Other methods

- int [erase](#) (size\_t ix)

## 3.269.2 Member Function Documentation

### 3.269.2.1 int free () [inline]

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 832 of file ovector\_tlate.h.

**3.269.2.2 int reserve (size\_t cap) [inline]**

Reserve memory by increasing capacity.

Increase the maximum capacity of the vector so that calls to [push\\_back\(\)](#) do not need to automatically increase the capacity.

This function quietly does nothing if `cap` is smaller than the present vector size given by [size\(\)](#).

Definition at line 914 of file `ovector_tlate.h`.

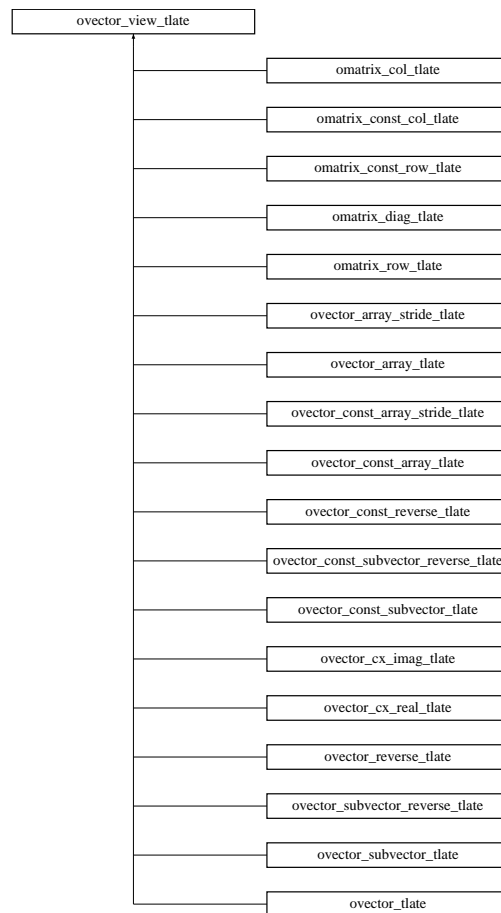
The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

**3.270 ovector\_view\_tlate Class Template Reference**

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ovector_view_tlate`:

**3.270.1 Detailed Description**

```
template<class data_t, class vparent_t, class block_t> class ovector_view_tlate< data_t, vparent_t, block_t >
```

A vector view with finite stride.

Definition at line 51 of file `ovector_tlate.h`.

## Public Member Functions

- [~ovector\\_view\\_tlate](#) ()

## Copy constructors

- [ovector\\_view\\_tlate](#) (const [ovector\\_view\\_tlate](#) &v)  
*So2scflow copy constructor - create a new view of the same vector.*
- [ovector\\_view\\_tlate](#) & [operator=](#) (const [ovector\\_view\\_tlate](#) &v)  
*So2scflow copy constructor - create a new view of the same vector.*
- [ovector\\_view\\_tlate](#) (const [uvector\\_view\\_tlate](#)< data\_t > &v)  
*So2scflow copy constructor - view a unit-stride vector.*
- [ovector\\_view\\_tlate](#) & [operator=](#) (const [uvector\\_view\\_tlate](#)< data\_t > &v)  
*So2scflow copy constructor - view a unit-stride vector.*

## Get and set methods

- data\_t & [operator\[\]](#) (size\_t i)  
*Array-like indexing.*
- const data\_t & [operator\[\]](#) (size\_t i) const  
*Array-like indexing.*
- data\_t & [operator\(\)](#) (size\_t i)  
*Array-like indexing.*
- const data\_t & [operator\(\)](#) (size\_t i) const  
*Array-like indexing.*
- data\_t [get](#) (size\_t i) const  
*Get (with optional range-checking).*
- data\_t \* [get\\_ptr](#) (size\_t i)  
*Get pointer (with optional range-checking).*
- const data\_t \* [get\\_const\\_ptr](#) (size\_t i) const  
*Get pointer (with optional range-checking).*
- int [set](#) (size\_t i, data\_t val)  
*Set (with optional range-checking).*
- int [set\\_all](#) (double val)  
*Set all of the value to be the value val.*
- size\_t [size](#) () const  
*Method to return vector size.*
- size\_t [capacity](#) () const  
*Method to return capacity.*
- size\_t [stride](#) () const  
*Method to return vector stride.*

## Arithmetic

- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator+=](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*operator+=*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator-=](#) (const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &x)  
*operator-=*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator+=](#) (data\_t &x)  
*operator+=*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator-=](#) (data\_t &x)  
*operator-=*
- [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > & [operator \\*=](#) (const data\_t &y)  
*operator\*=*
- data\_t [norm](#) () const  
*Norm.*

## Other methods

- int [swap](#) (ovector\_view\_tlate< data\_t, vparent\_t, block\_t > &x)  
*Swap vectors.*
- bool [is\\_owner](#) () const  
*Return true if this object owns the data it refers to.*
- size\_t [lookup](#) (const data\_t x0) const  
*Exhaustively look through the array for a particular value.*
- data\_t [max](#) () const  
*Find the maximum element.*
- size\_t [max\\_index](#) () const  
*Find the location of the maximum element.*
- data\_t [min](#) () const  
*Find the minimum element.*
- size\_t [min\\_index](#) () const  
*Find the location of the minimum element.*
- vparent\_t \* [get\\_gsl\\_vector](#) ()  
*Return a gsl vector.*
- const vparent\_t \* [get\\_gsl\\_vector\\_const](#) () const  
*Return a const gsl vector.*

### Protected Member Functions

- [ovector\\_view\\_tlate](#) ()  
*Empty constructor provided for use by ovector\_tlate(const ovector\_tlate &v).*

## 3.270.2 Member Function Documentation

### 3.270.2.1 size\_t size () const [inline]

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 237 of file ovector\_tlate.h.

### 3.270.2.2 size\_t capacity () const [inline]

Method to return capacity.

Analogous to `std::vector<>.capacity()`

Definition at line 246 of file ovector\_tlate.h.

### 3.270.2.3 size\_t stride () const [inline]

Method to return vector stride.

If no memory has been allocated, this will quietly return zero.

Definition at line 257 of file ovector\_tlate.h.

### 3.270.2.4 bool is\_owner () const [inline]

Return true if this object owns the data it refers to.

This can be used to determine if an object is a "vector\_view", or a legitimate "vector". If [is\\_owner\(\)](#) is true, then it is an [ovector\\_tlate](#) object.

If any O2scl class creates a [ovector\\_tlate](#) object in which [is\\_owner\(\)](#) returns false, then it is a bug!

Definition at line 353 of file ovector\_tlate.h.

**3.270.2.5 size\_t lookup (const data\_t x0) const** [inline]

Exhaustively look through the array for a particular value.

This can only fail if *all* of the entries in the array are not finite, in which case it calls [set\\_err\(\)](#) and returns 0. The error handler is reset at the beginning of [lookup\(\)](#).

Definition at line 365 of file [ovector\\_tlate.h](#).

The documentation for this class was generated from the following file:

- [ovector\\_tlate.h](#)

**3.271 pinside Class Reference**

```
#include <pinside.h>
```

**3.271.1 Detailed Description**

Test [line](#) intersection and [point](#) inside polygon.

This is a fast and dirty implementation of the [point](#) inside polygon test from Jerome L. Lewis, SIGSCE Bulletin, 34 (2002) 81.

Note that an error in that article ("count-" should have been "count-") has been corrected here.

Definition at line 45 of file [pinside.h](#).

**Public Member Functions**

- int [intersect](#) (double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)  
*Determine if two [line](#) segments intersect.*
- int [inside](#) (double x, double y, const [o2scl::ovector\\_view](#) &xa, const [o2scl::ovector\\_view](#) &ya)  
*Determine if [point](#) (x,y) is inside a polygon.*
- int [test](#) ([test\\_mgr](#) &t)  
*Perform some simple testing.*

**Protected Member Functions**

- int [intersect](#) ([line](#) P, [line](#) Q)  
*Test if [line](#) segments P and Q intersect.*
- int [inside](#) ([point](#) t, [point](#) p[ ], int N)  
*Test if [point](#) t is inside polygon p of size N.*

**Data Structures**

- struct [line](#)  
*Internal [line](#) definition.*
- struct [point](#)  
*Internal [point](#) definition.*

**3.271.2 Member Function Documentation****3.271.2.1 int intersect (double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)** [inline]

Determine if two [line](#) segments intersect.

---



The function returns 1 if the [line](#) segment determined by the endpoints  $(x_1, y_1)$  and  $(x_2, y_2)$  and the [line](#) segment determined by the endpoints  $(x_3, y_3)$  and  $(x_4, y_4)$  intersect, and 0 otherwise.

Definition at line 78 of file pinside.h.

### 3.271.2.2 int inside (double x, double y, const o2scl::ovector\_view & xa, const o2scl::ovector\_view & ya)

Determine if [point](#) (x,y) is inside a polygon.

This returns 1 if the [point](#) (x,y) is inside the polygon defined by xa and ya, and 0 otherwise.

Note that if the [point](#) (x,y) is exactly on the polygon, then the result of this function is not well-defined and it will return either 0 or 1.

The documentation for this class was generated from the following file:

- pinside.h

## 3.272 pinside::line Struct Reference

```
#include <pinside.h>
```

### 3.272.1 Detailed Description

Internal [line](#) definition.

Definition at line 56 of file pinside.h.

#### Data Fields

- [point](#) p1
- [point](#) p2

The documentation for this struct was generated from the following file:

- pinside.h

## 3.273 pinside::point Struct Reference

```
#include <pinside.h>
```

### 3.273.1 Detailed Description

Internal [point](#) definition.

Definition at line 50 of file pinside.h.

#### Data Fields

- double x
- double y

The documentation for this struct was generated from the following file:

- pinside.h
-

## 3.274 planar\_intp Class Template Reference

```
#include <planar_intp.h>
```

### 3.274.1 Detailed Description

**template<class vec\_t, class mat\_t> class planar\_intp< vec\_t, mat\_t >**

Interpolate among two independent variables with planes.

This is an analog of 1-d linear interpolation for two dimensions. For a set of data  $x_i, y_i, f_{j,i}$ , values for  $f_j$  are predicted given a value of  $x$  and  $y$ . (In contrast to [twod\\_intp](#), the data need not be presented in a grid.) This is done by finding the plane that goes through three closest points in the data set.

This procedure will fail if the three closest points are co-linear, and [interp\(\)](#) will then call [set\\_err\(\)](#) and return zero.

There is no caching so the numeric values of the data may be freely changed between calls to [interp\(\)](#).

The vector and matrix types can be any types which have suitably defined functions `operator [ ]`.

#### Idea for future

Rewrite so that it never fails unless all the points in the data set lie on a line. This would probably demand sorting all of the points by distance from desired location.

Definition at line 60 of file `planar_intp.h`.

### Public Member Functions

- [planar\\_intp](#) ()
- int [set\\_data](#) (size\_t n\_points, vec\_t &x, vec\_t &y, size\_t n\_dat, mat\_t &dat)  
*Initialize the data for the planar interpolation.*
- int [interp](#) (double x, double y, vec\_t &ip)  
*Perform the planar interpolation.*
- int [interp](#) (double x, double y, vec\_t &ip, double &x1, double &y1, double &x2, double &y2, double &x3, double &y3)  
*Planar interpolation returning the closest points.*

### Protected Member Functions

- int [swap](#) (int &i1, double &c1, int &i2, double &c2)  
*Swap points 1 and 2.*

### Protected Attributes

- size\_t [np](#)  
*The number of points.*
- size\_t [nd](#)  
*The number of functions.*
- vec\_t \* [ux](#)  
*The x-values.*
- vec\_t \* [uy](#)  
*The y-values.*
- mat\_t \* [udat](#)  
*The data.*
- bool [data\\_set](#)  
*True if the data has been specified.*

### 3.274.2 Member Function Documentation

#### 3.274.2.1 `int interp (double x, double y, vec_t & ip)` `[inline]`

Perform the planar interpolation.

It is assumed that `ip` is properly allocated beforehand.

Definition at line 94 of file `planar_intp.h`.

#### 3.274.2.2 `int interp (double x, double y, vec_t & ip, double & x1, double & y1, double & x2, double & y2, double & x3, double & y3)` `[inline]`

Planar interpolation returning the closest points.

This function interpolates `x` and `y` into the data returning `ip`. It also returns the three closest `x`- and `y`-values used for computing the plane.

It is assumed that `ip` is properly allocated beforehand.

Put in initial points

Sort initial points

Definition at line 108 of file `planar_intp.h`.

The documentation for this class was generated from the following file:

- `planar_intp.h`

## 3.275 `pointer_alloc` Class Template Reference

```
#include <array.h>
```

### 3.275.1 Detailed Description

`template<class base_t> class pointer_alloc< base_t >`

A simple class to provide an `allocate()` function for pointers.

Uses `new` and `delete`.

Definition at line 107 of file `array.h`.

#### Public Member Functions

- void `allocate` (`base_t *&v`, `int i`)  
*Allocate `v` for `i` elements.*
- void `free` (`base_t *&v`)  
*Free memory.*

The documentation for this class was generated from the following file:

- `array.h`

## 3.276 `pointer_input` Struct Reference

```
#include <collection.h>
```

---

### 3.276.1 Detailed Description

A pointer input structure.

Definition at line 76 of file collection.h.

#### Data Fields

- std::string [name](#)  
*The name of the pointer.*
- void \*\* [ptr](#)  
*The pointer.*
- std::string [stype](#)  
*The type of the object pointed to.*

The documentation for this struct was generated from the following file:

- collection.h

## 3.277 **pointer\_output** Struct Reference

```
#include <collection.h>
```

### 3.277.1 Detailed Description

A pointer output structure.

Definition at line 66 of file collection.h.

#### Data Fields

- std::string [name](#)  
*The name of the pointer.*
- [collection\\_entry](#) \* [ep](#)  
*Pointer to the [collection](#) entry.*
- bool [output](#)  
*True if the pointer has been written to the file.*

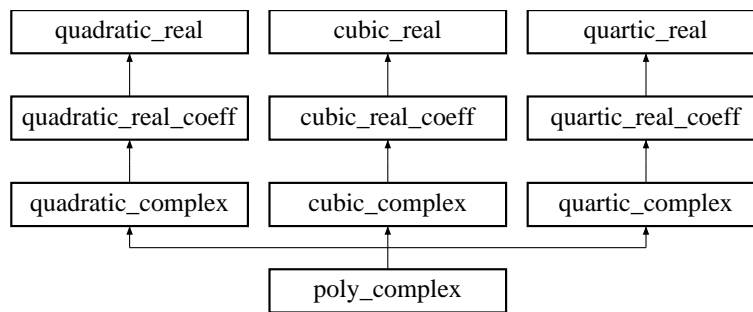
The documentation for this struct was generated from the following file:

- collection.h

## 3.278 **poly\_complex** Class Reference

```
#include <poly.h>
```

Inheritance diagram for poly\_complex::



### 3.278.1 Detailed Description

Base class for solving a general polynomial with complex coefficients.

Definition at line 362 of file poly.h.

#### Public Member Functions

- virtual [~poly\\_complex](#) ()
- virtual int [solve\\_c](#) (int n, const std::complex< double > co[], std::complex< double > ro[])  
*Solve the n-th order polynomial.*
- virtual int [polish\\_c](#) (int n, const std::complex< double > co[], std::complex< double > \*ro)  
*Polish the roots.*
- const char \* [type](#) ()  
*Return a string denoting the type ("poly\_complex").*

### 3.278.2 Member Function Documentation

**3.278.2.1** virtual int [solve\\_c](#) (int n, const std::complex< double > co[], std::complex< double > ro[]) [inline, virtual]

Solve the n-th order polynomial.

The coefficients are stored in co[], with the leading coefficient as co[0] and the constant term as co[n]. The roots are returned in ro[0],...,ro[n-1].

Definition at line 376 of file poly.h.

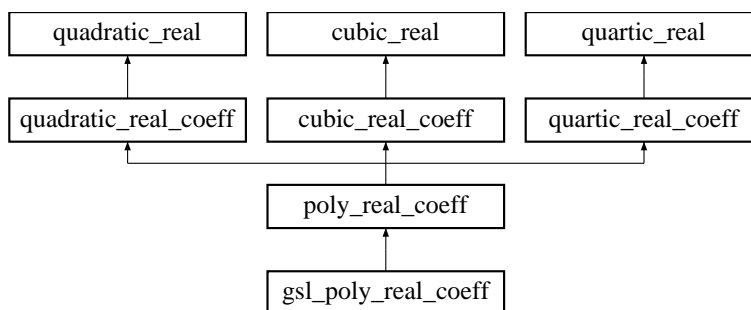
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.279 poly\_real\_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for poly\_real\_coeff::



### 3.279.1 Detailed Description

Base class for solving a general polynomial with real coefficients and complex roots.

Definition at line 334 of file poly.h.

#### Public Member Functions

- virtual [~poly\\_real\\_coeff](#) ()
- virtual int [solve\\_rc](#) (int n, const double co[], std::complex< double > ro[])  
*Solve the n-th order polynomial.*
- virtual int [polish\\_rc](#) (int n, const double co[], std::complex< double > \*ro)  
*Polish the roots.*
- const char \* [type](#) ()  
*Return a string denoting the type ("poly\_real\_coeff").*

### 3.279.2 Member Function Documentation

#### 3.279.2.1 virtual int solve\_rc (int n, const double co[], std::complex< double > ro[]) [inline, virtual]

Solve the n-th order polynomial.

The coefficients are stored in co[], with the leading coefficient as co[0] and the constant term as co[n]. The roots are returned in ro[0],...,ro[n-1].

Reimplemented in [gsl\\_poly\\_real\\_coeff](#).

Definition at line 348 of file poly.h.

The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.280 polylog Class Reference

```
#include <polylog.h>
```

### 3.280.1 Detailed Description

Polylogarithms (approximate)  $Li_n(x)$ .

This gives an approximation to the polylogarithm functions.

Only works at present for  $n = 0, 1, \dots, 6$ . Uses GSL library for  $n=2$ .

Uses linear interpolation for  $-1 < x < 0$  and a series expansion for  $x < -1$

**Todo**

- Give error estimate?
- Improve accuracy?
- Use more sophisticated interpolation?
- Add the series  $Li(n, x) = x + 2^{-n}x^2 + 3^{-n}x^3 + \dots$  for  $x \rightarrow 0$ ?
- Implement for positive arguments  $< 1.0$
- Make another [polylog](#) class which implements series acceleration?

For reference, there are exact relations

$$\begin{aligned} \text{Li}_2\left(\frac{1}{2}\right) &= \frac{1}{12} \left[ \pi^2 - 6 (\ln 2)^2 \right] \\ \text{Li}_3\left(\frac{1}{2}\right) &= \frac{1}{24} \left[ 4 (\ln 2)^3 - 2\pi^2 \ln 2 + 21\zeta(3) \right] \\ \text{Li}_{-1}(x) &= \frac{x}{(1-x)^2} \\ \text{Li}_{-2}(x) &= \frac{x(x+1)}{(1-x)^3} \end{aligned}$$

Definition at line 77 of file polylog.h.

**Public Member Functions**

- double [li0](#) (double x)  
*0-th order polylogarithm*  $= x/(1-x)$
- double [li1](#) (double x)  
*1-st order polylogarithm*  $= -\ln(1-x)$
- double [li2](#) (double x)  
*2-nd order polylogarithm*
- double [li3](#) (double x)  
*3-rd order polylogarithm*
- double [li4](#) (double x)  
*4-th order polylogarithm*
- double [li5](#) (double x)  
*5-th order polylogarithm*
- double [li6](#) (double x)  
*6-th order polylogarithm*

The documentation for this class was generated from the following file:

- polylog.h

**3.281 quad\_intp Class Template Reference**

```
#include <quad_intp.h>
```

### 3.281.1 Detailed Description

**template<class vec\_t, class mat\_t> class quad\_intp< vec\_t, mat\_t >**

Interpolate a function of two independent variables with a quadratic polynomial.

This is a "conic-section" interpolation for two dimensions, using the function

$$z(x, y) = a_1x^2 + a_2xy + a_3y^2 + a_4x + a_5y + a_6$$

For a set of data  $x_i, y_i, z_i$ , a value of  $z$  is predicted given a value of  $x$  and  $y$ . This is done by finding the conic section obeying the above relation that goes through six closest points in the data set.

This procedure does not always succeed. It fails when the 6 closest points are somehow degenerate, for example, if they are all colinear.

The vector and matrix types can be any types which have suitably defined functions `operator[]`.

There is no caching so the numeric values of the data may be freely changed between calls to `interp()`.

#### Bug

This class doesn't seem to work at present.

Definition at line 64 of file `quad_intp.h`.

#### Public Member Functions

- `quad_intp()`
- `int compare` (const void \*x, const void \*y)
- `int set_data` (size\_t n\_points, vec\_t &x, vec\_t &y, size\_t n\_dat, mat\_t &dat)  
*Initialize the data for the quad interpolation.*
- `int interp` (double x, double y, vec\_t &ip)  
*Perform the quadratic interpolation.*

#### Protected Member Functions

- `int swap` (int &i1, double &c1, int &i2, double &c2)

#### Protected Attributes

- `int np`  
*The number of grid points.*
- `int nd`  
*The number of functions.*
- `vec_t * ux`  
*The x-values.*
- `vec_t * uy`  
*The y-values.*
- `mat_t * udat`  
*The data.*
- `bool data_set`  
*True if the data has been given by the user.*

#### Data Structures

- `struct point`



### 3.281.2 Member Function Documentation

#### 3.281.2.1 int interp (double x, double y, vec\_t & ip) [inline]

Perform the quadratic interpolation.

It is assumed that `ip` is properly allocated beforehand.

Definition at line 105 of file `quad_intp.h`.

The documentation for this class was generated from the following file:

- `quad_intp.h`

## 3.282 quad\_intp::point Struct Reference

### 3.282.1 Detailed Description

**template<class vec\_t, class mat\_t> struct quad\_intp< vec\_t, mat\_t >::point**

Definition at line 71 of file `quad_intp.h`.

#### Data Fields

- int `i`
- double `c`

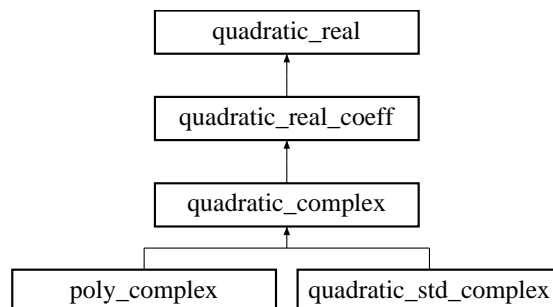
The documentation for this struct was generated from the following file:

- `quad_intp.h`

## 3.283 quadratic\_complex Class Reference

```
#include <poly.h>
```

Inheritance diagram for `quadratic_complex::`



### 3.283.1 Detailed Description

Solve a quadratic polynomial with complex coefficients and complex roots.

Definition at line 105 of file `poly.h`.

## Public Member Functions

- virtual `~quadratic_complex()`
- virtual int `solve_r`(const double a2, const double b2, const double c2, double &x1, double &x2)  
Solves the polynomial  $a_2x^2 + b_2x + c_2 = 0$  giving the two solutions  $x = x_1$  and  $x = x_2$ .
- virtual int `solve_rc`(const double a2, const double b2, const double c2, std::complex< double > &x1, std::complex< double > &x2)  
Solves the polynomial  $a_2x^2 + b_2x + c_2 = 0$  giving the two complex solutions  $x = x_1$  and  $x = x_2$ .
- virtual int `solve_c`(const std::complex< double > a2, const std::complex< double > b2, const std::complex< double > c2, std::complex< double > &x1, std::complex< double > &x2)  
Solves the complex polynomial  $a_2x^2 + b_2x + c_2 = 0$  giving the two complex solutions  $x = x_1$  and  $x = x_2$ .
- const char \* `type()`  
Return a string denoting the type ("quadratic\_complex").

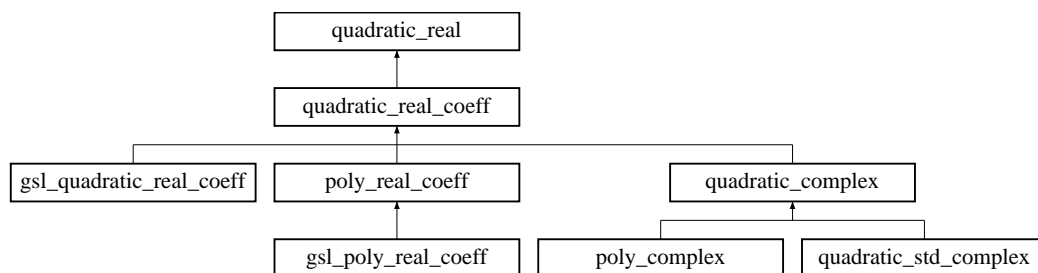
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.284 quadratic\_real Class Reference

```
#include <poly.h>
```

Inheritance diagram for quadratic\_real::



### 3.284.1 Detailed Description

Solve a quadratic polynomial with real coefficients and real roots.

Definition at line 59 of file `poly.h`.

## Public Member Functions

- virtual `~quadratic_real()`
- virtual int `solve_r`(const double a2, const double b2, const double c2, double &x1, double &x2)  
Solves the polynomial  $a_2x^2 + b_2x + c_2 = 0$  giving the two solutions  $x = x_1$  and  $x = x_2$ .
- const char \* `type()`  
Return a string denoting the type ("quadratic\_real").

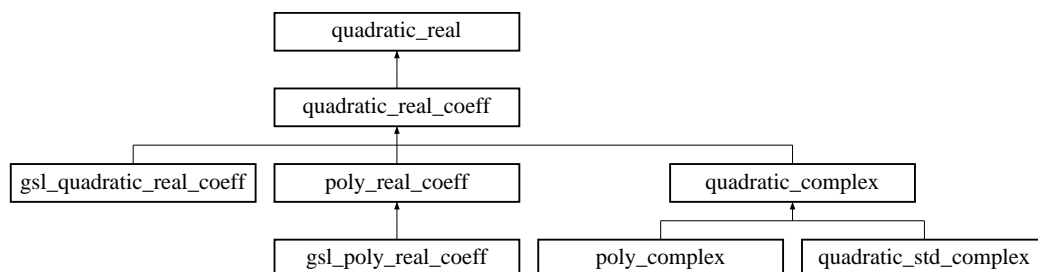
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.285 quadratic\_real\_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for quadratic\_real\_coeff::



### 3.285.1 Detailed Description

Solve a quadratic polynomial with real coefficients and complex roots.

Definition at line 78 of file poly.h.

#### Public Member Functions

- virtual [~quadratic\\_real\\_coeff](#) ()
- virtual int [solve\\_r](#) (const double a2, const double b2, const double c2, double &x1, double &x2)  
Solves the polynomial  $a_2x^2 + b_2x + c_2 = 0$  giving the two solutions  $x = x_1$  and  $x = x_2$ .
- virtual int [solve\\_rc](#) (const double a2, const double b2, const double c2, std::complex< double > &x1, std::complex< double > &x2)  
Solves the polynomial  $a_2x^2 + b_2x + c_2 = 0$  giving the two complex solutions  $x = x_1$  and  $x = x_2$ .
- const char \* [type](#) ()  
Return a string denoting the type ("quadratic\_real\_coeff").

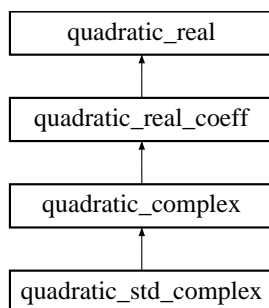
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.286 quadratic\_std\_complex Class Reference

```
#include <poly.h>
```

Inheritance diagram for quadratic\_std\_complex::



### 3.286.1 Detailed Description

Solve a quadratic with complex coefficients and complex roots.

Definition at line 592 of file `poly.h`.

#### Public Member Functions

- virtual `~quadratic_std_complex` ()
- virtual int `solve_c` (const std::complex< double > a2, const std::complex< double > b2, const std::complex< double > c2, std::complex< double > &x1, std::complex< double > &x2)  
*Solves the complex polynomial  $a_2x^2 + b_2x + c_2 = 0$  giving the two complex solutions  $x = x_1$  and  $x = x_2$ .*
- const char \* `type` ()  
*Return a string denoting the type ("quadratic\_std\_complex").*

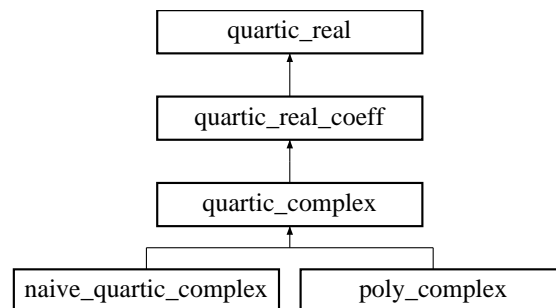
The documentation for this class was generated from the following file:

- `poly.h`

## 3.287 `quartic_complex` Class Reference

```
#include <poly.h>
```

Inheritance diagram for `quartic_complex`:



### 3.287.1 Detailed Description

Solve a quartic polynomial with complex coefficients and complex roots.

Definition at line 287 of file `poly.h`.

#### Public Member Functions

- virtual `~quartic_complex` ()
- virtual int `solve_r` (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)
- virtual int `solve_rc` (const double a4, const double b4, const double c4, const double d4, const double e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)
- virtual int `solve_c` (const std::complex< double > a4, const std::complex< double > b4, const std::complex< double > c4, const std::complex< double > d4, const std::complex< double > e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)
- const char \* `type` ()  
*Return a string denoting the type ("quartic\_complex").*

### 3.287.2 Member Function Documentation

**3.287.2.1** `virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double & x1, double & x2, double & x3, double & x4) [virtual]`

Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented from [quartic\\_real\\_coeff](#).

**3.287.2.2** `virtual int solve_rc (const double a4, const double b4, const double c4, const double d4, const double e4, std::complex< double > & x1, std::complex< double > & x2, std::complex< double > & x3, std::complex< double > & x4) [virtual]`

Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four complex solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented from [quartic\\_real\\_coeff](#).

**3.287.2.3** `virtual int solve_c (const std::complex< double > a4, const std::complex< double > b4, const std::complex< double > c4, const std::complex< double > d4, const std::complex< double > e4, std::complex< double > & x1, std::complex< double > & x2, std::complex< double > & x3, std::complex< double > & x4) [inline, virtual]`

Solves the complex polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four complex solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented in [naive\\_quartic\\_complex](#).

Definition at line 318 of file `poly.h`.

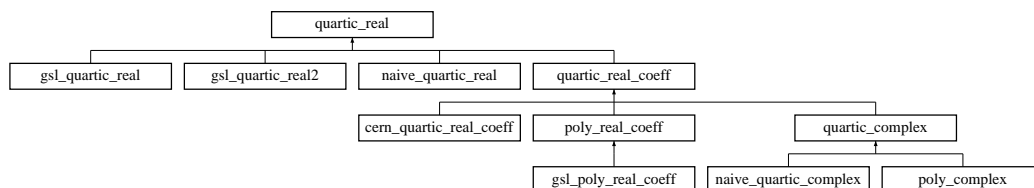
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.288 `quartic_real` Class Reference

```
#include <poly.h>
```

Inheritance diagram for `quartic_real`:



### 3.288.1 Detailed Description

Solve a quartic polynomial with real coefficients and real roots.

Definition at line 229 of file `poly.h`.

#### Public Member Functions

- `virtual ~quartic_real ()`
- `virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)`

- `const char * type ()`  
Return a string denoting the type ("quartic\_real").

### 3.288.2 Member Function Documentation

**3.288.2.1** `virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double & x1, double & x2, double & x3, double & x4)` [`inline`, `virtual`]

Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented in [quartic\\_real\\_coeff](#), [quartic\\_complex](#), [gsl\\_quartic\\_real](#), [gsl\\_quartic\\_real2](#), and [naive\\_quartic\\_real](#).

Definition at line 240 of file `poly.h`.

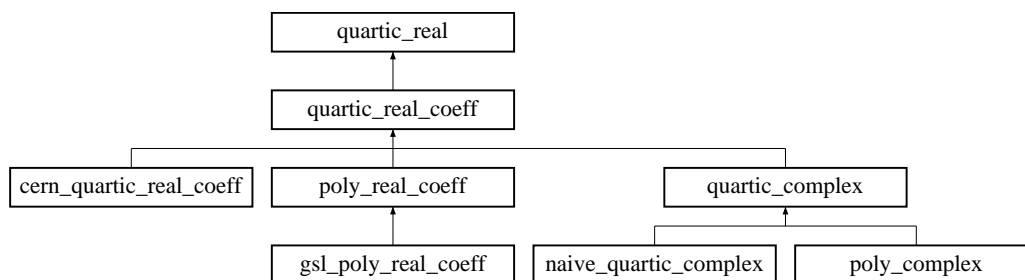
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.289 `quartic_real_coeff` Class Reference

```
#include <poly.h>
```

Inheritance diagram for `quartic_real_coeff`:



### 3.289.1 Detailed Description

Solve a quartic polynomial with real coefficients and complex roots.

Definition at line 253 of file `poly.h`.

#### Public Member Functions

- `virtual ~quartic_real_coeff ()`
- `virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)`
- `virtual int solve_rc (const double a4, const double b4, const double c4, const double d4, const double e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)`
- `const char * type ()`  
Return a string denoting the type ("quartic\_real\_coeff").

### 3.289.2 Member Function Documentation

**3.289.2.1** `virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double & x1, double & x2, double & x3, double & x4)` [`virtual`]

Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented from [quartic\\_real](#).

Reimplemented in [quartic\\_complex](#).

**3.289.2.2** `virtual int solve_rc (const double a4, const double b4, const double c4, const double d4, const double e4, std::complex< double > & x1, std::complex< double > & x2, std::complex< double > & x3, std::complex< double > & x4) [inline, virtual]`

Solves the polynomial  $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$  giving the four complex solutions  $x = x_1$ ,  $x = x_2$ ,  $x = x_3$ , and  $x = x_4$ .

Reimplemented in [quartic\\_complex](#), [cern\\_quartic\\_real\\_coeff](#), and [gsl\\_poly\\_real\\_coeff](#).

Definition at line 273 of file poly.h.

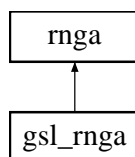
The documentation for this class was generated from the following file:

- [poly.h](#)

## 3.290 rnga Class Reference

```
#include <rnga.h>
```

Inheritance diagram for rnga::



### 3.290.1 Detailed Description

Random number generator base.

Using virtual functions is not recommended for random number generators, as speed is often an important issue. In order to facilitate the use of templates for routines which require random number generators, all descendants ought to provide the following functions:

- `double random()` - Provide a random number in [0.0,1.0)
- `unsigned long int random_int(unsigned long int n)` - Provide a random integer in [0,n-1]
- `unsigned long int get_max()` - Return the maximum integer for the random number generator. The argument to `random_int()` should be less than the value returned by `get_max()`.

Definition at line 50 of file rnga.h.

### Public Member Functions

- [rnga\(\)](#)
- void [clock\\_seed\(\)](#)  
*Initialize the seed with a value taken from the computer clock.*
- unsigned long int [get\\_seed\(\)](#)  
*Get the seed.*
- void [set\\_seed](#) (unsigned long int s)  
*Set the seed.*

## Protected Member Functions

- `rnga` (const `rnga` &)
- `rnga & operator=` (const `rnga` &)

## Protected Attributes

- unsigned long int `seed`  
*The seed.*

## 3.290.2 Member Function Documentation

### 3.290.2.1 void clock\_seed ()

Initialize the seed with a value taken from the computer clock.

This is a naive seed generator which uses `seed=time(NULL)` to generate a seed.

### Todo

Ensure this function is ANSI compatible

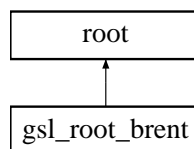
The documentation for this class was generated from the following file:

- `rnga.h`

## 3.291 root Class Template Reference

```
#include <root.h>
```

Inheritance diagram for `root::`



### 3.291.1 Detailed Description

```
template<class param_t, class func_t, class dfunc_t = func_t> class root< param_t, func_t, dfunc_t >
```

1-dimensional solver base class

#### Note:

This class does not actually do any solving, it is present to provide member data and various functions common to all the 1 dimensional solvers.

Definition at line 45 of file `root.h`.



## Public Member Functions

- `root()`
- `virtual ~root()`
- `virtual const char * type()`  
*Return the type, "root".*
- `virtual int print_iter(double x, double y, int iter, double value=0.0, double limit=0.0, std::string comment="")`  
*Print out iteration information.*
- `virtual int solve(double &x, param_t &pa, func_t &func)`  
*Solve func using x as an initial guess.*
- `virtual int solve_bkt(double &x1, double x2, param_t &pa, func_t &func)`  
*Solve func in region  $x_1 < x < x_2$  returning  $x_1$ .*
- `virtual int solve_de(double &x, param_t &pa, func_t &func, dfunc_t &df)`  
*Solve func using x as an initial guess using derivatives df.*

## Data Fields

- `double tolf`  
*The maximum value of the functions for success (default  $10^{-8}$ ).*
- `double tolx`  
*The minimum allowable stepsize (default  $10^{-12}$ ).*
- `int verbose`  
*Output control (default 0).*
- `int ntrial`  
*Maximum number of iterations (default 100).*
- `bool over_bkt`  
*Should be true if root\_bkt() is overloaded.*
- `bool over_de`  
*Should be true if root\_de() is overloaded.*
- `double deriv_eps`  
*The stepsize for finite-differencing (default 0).*
- `int last_ntrial`  
*The number of iterations for in the most recent minimization.*

### 3.291.2 Member Function Documentation

**3.291.2.1** `virtual int print_iter(double x, double y, int iter, double value = 0.0, double limit = 0.0, std::string comment = "")` [inline, virtual]

Print out iteration information.

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character before continuing

Definition at line 112 of file root.h.

### 3.291.3 Field Documentation

#### 3.291.3.1 double deriv\_eps

The stepsize for finite-differencing (default 0).

If this is zero, then  $10^{-4}$  times the argument will be used.

#### Note:

It seems this variable is left over from an earlier version of the code. I'm keeping it here for now just in case.

Definition at line 92 of file root.h.

The documentation for this class was generated from the following file:

- root.h

## 3.292 search\_vec Class Template Reference

```
#include <search_vec.h>
```

### 3.292.1 Detailed Description

**template<class vec\_t> class search\_vec< vec\_t >**

Searching class for monotonic data.

A searching class for monotonic vectors. A caching system similar to `gsl_interp_accel` is used.

For normal usage, just use `find()`. If you happen to know in advance that the vector is increasing or decreasing, then you can use `find_inc()` or `find_dec()` instead.

#### Todo

The documentation here is still kind of unclear.

Definition at line 52 of file search\_vec.h.

### Public Member Functions

- `size_t find` (const double x0, size\_t n, const vec\_t &x)  
*Search an increasing or decreasing vector.*
- `size_t find_inc` (const double x0, size\_t n, const vec\_t &x)  
*Search part of a increasing vector.*
- `size_t find_dec` (const double x0, size\_t n, const vec\_t &x)  
*Search part of a decreasing vector.*
- `size_t ordered_lookup` (const double x0, size\_t n, const vec\_t &x)  
*Find the index of x0 in the ordered array x.*
- `size_t ordered_interval` (const double x0, size\_t n, const vec\_t &x)  
*Find the interval containing x0 in the ordered array x.*
- `size_t bsearch_inc` (const double x0, const vec\_t &x, size\_t lo, size\_t hi) const  
*Binary search a part of an increasing vector.*
- `size_t bsearch_dec` (const double x0, const vec\_t &x, size\_t lo, size\_t hi) const  
*Binary search a part of an decreasing vector.*

### Protected Attributes

- `size_t cache`  
*Storage for the most recent index.*

### 3.292.2 Member Function Documentation

#### 3.292.2.1 size\_t ordered\_lookup (const double x0, size\_t n, const vec\_t &x) [inline]

Find the index of x0 in the ordered array x.

This returns the index  $i$  for which  $x[i]$  is as close as possible to  $x_0$  if  $x[i]$  is either increasing or decreasing.

If some of the values in the ovector are not finite, then the output of this function is not defined.

If  $x[i]$  is non-monotonic, consider using [ovector\\_view\\_tlate::lookup\(\)](#) or [uvector\\_view\\_tlate::lookup\(\)](#) instead of this function.

Definition at line 120 of file `search_vec.h`.

### 3.292.2.2 `size_t ordered_interval (const double x0, size_t n, const vec_t & x)` [inline]

Find the interval containing  $x_0$  in the ordered array  $x$ .

This returns the index  $i$  for which  $x[i] \leq x_0 < x[i+1]$ .

If the array is increasing and  $x_0 < x[0]$ , then 0 is returned. If the array is increasing and  $x_0 > x[n-1]$ , then  $nvar-1$  is returned (this behavior is slightly different from GSL). The decreasing case is handled analogously.

If some of the values in the vector are not finite, then the output of this function is not defined.

If  $x[i]$  is non-monotonic, consider using [ovector\\_view\\_tlate::lookup\(\)](#) or [uvector\\_view\\_tlate::lookup\(\)](#) instead of this function.

Definition at line 168 of file `search_vec.h`.

### 3.292.2.3 `size_t bsearch_inc (const double x0, const vec_t & x, size_t lo, size_t hi) const` [inline]

Binary search a part of an increasing vector.

The cache is not used for this function.

Definition at line 194 of file `search_vec.h`.

### 3.292.2.4 `size_t bsearch_dec (const double x0, const vec_t & x, size_t lo, size_t hi) const` [inline]

Binary search a part of an decreasing vector.

The cache is not used for this function.

Definition at line 212 of file `search_vec.h`.

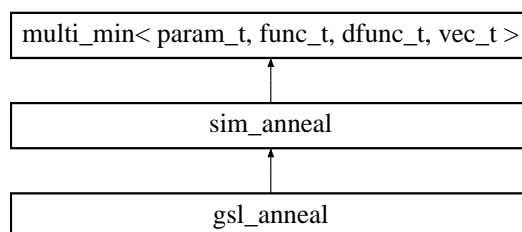
The documentation for this class was generated from the following file:

- `search_vec.h`

## 3.293 **sim\_anneal Class Template Reference**

```
#include <sim_anneal.h>
```

Inheritance diagram for `sim_anneal::`



### 3.293.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t = ovector\_view, class rng\_t = gsl\_rng> class sim\_anneal< param\_t, func\_t, vec\_t, rng\_t >**

Simulated annealing base.

The seed of the generator is not fixed initially by calls to [mmin\(\)](#), so if successive calls should reproduce the same results, then the random seed should be set by the user before each call.

For the algorithms here, it is important that all of the inputs  $x[i]$  to the function are scaled similarly relative to the temperature. For example, if the inputs  $x[i]$  are all of order 1, one might consider a temperature schedule which begins with  $T = 1$ .

The number of iterations at each temperature is controlled by [minimize::ntrial](#) which defaults to 100.

Definition at line 58 of file `sim_anneal.h`.

#### Public Member Functions

- [sim\\_anneal\(\)](#)
- virtual [~sim\\_anneal\(\)](#)
- virtual int [mmin](#) (size\_t nvar, vec\_t &x, double &fmin, param\_t &pa, func\_t &func)  
*Calculate the minimum fmin of func w.r.t the array x of size nvar.*
- int [set\\_tptr\\_schedule](#) (tptr\_schedule< vec\_t > &tr)  
*Specify the temperature schedule.*
- virtual int [print\\_iter](#) (size\_t nv, vec\_t &x, double y, int iter, double tptr, std::string comment)  
*Print out iteration information.*
- virtual const char \* [type](#) ()  
*Return string denoting type, "sim\_anneal".*

#### Data Fields

- rng\_t [def\\_rng](#)  
*The default random number generator.*
- tptr\_geoseries< vec\_t > [def\\_schedule](#)  
*The default temperature schedule.*

#### Protected Attributes

- tptr\_schedule< vec\_t > \* [tp](#)  
*Pointer to the temperature annealing schedule.*

### 3.293.2 Member Function Documentation

**3.293.2.1 virtual int print\_iter (size\_t nv, vec\_t & x, double y, int iter, double tptr, std::string comment)** [[inline](#), [virtual](#)]

Print out iteration information.

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to `std::cout` along with the iteration number. If verbose>=2 then each iteration waits for a character.

Definition at line 103 of file `sim_anneal.h`.

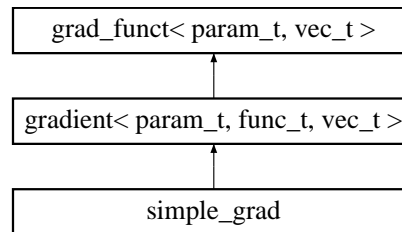
The documentation for this class was generated from the following file:

- `sim_anneal.h`

## 3.294 simple\_grad Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for simple\_grad::



### 3.294.1 Detailed Description

**template<class param\_t, class func\_t, class vec\_t> class simple\_grad< param\_t, func\_t, vec\_t >**

Simple automatic computation of [gradient](#) by finite differencing.

Definition at line 168 of file multi\_min.h.

#### Public Member Functions

- [simple\\_grad](#) ()
- virtual [~simple\\_grad](#) ()
- virtual int [operator\(\)](#) (size\_t nv, vec\_t &x, vec\_t &g, param\_t &pa)  
*Compute the [gradient](#) g at the point x.*

#### Data Fields

- double [epsrel](#)  
*The relative stepsize for finite-differencing (default  $10^{-4}$ ).*
- double [epsmin](#)  
*The minimum stepsize (default  $10^{-15}$ ).*

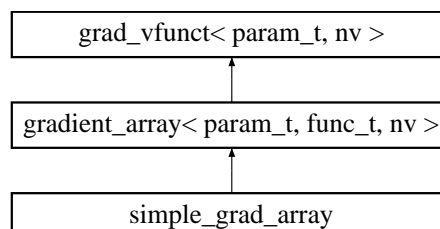
The documentation for this class was generated from the following file:

- multi\_min.h

## 3.295 simple\_grad\_array Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for simple\_grad\_array::



### 3.295.1 Detailed Description

`template<class param_t, class func_t, size_t nv> class simple_grad_array< param_t, func_t, nv >`

Simple automatic computation of [gradient](#) by finite differencing with arrays.

Definition at line 352 of file multi\_min.h.

#### Public Member Functions

- [simple\\_grad\\_array](#) ()
- virtual [~simple\\_grad\\_array](#) ()
- virtual int [operator\(\)](#) (size\_t nvar, double x[nv], double g[nv], param\_t &pa)  
Compute the [gradient](#)  $g$  at the point  $x$ .

#### Data Fields

- double [epsrel](#)  
The relative stepsize for finite-differencing (default  $10^{-4}$ ).
- double [epsmin](#)  
The minimum stepsize (default  $10^{-15}$ ).

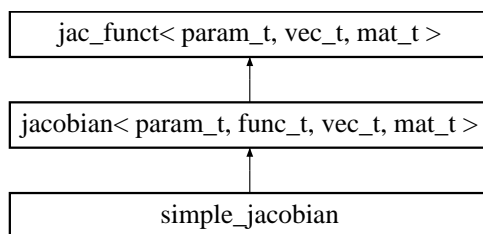
The documentation for this class was generated from the following file:

- multi\_min.h

## 3.296 simple\_jacobian Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for simple\_jacobian::



### 3.296.1 Detailed Description

`template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class simple_jacobian< param_t, func_t, vec_t, mat_t, alloc_vec_t, alloc_t >`

Simple automatic Jacobian.

This simple routine is nearly equivalent to GSL as given in `multiroots/fdjac.c`. It has an additional test to ensure that the finite-differencing stepsize does not vanish, and returns an error if the Jacobian is singular. To obtain the GSL behavior, set [epsrel](#) to `GSL_SQRT_DBL_EPSILON` and set [epsmin](#) to zero.

This class does not separately check the vector and matrix sizes to ensure they are commensurate.

## Todo

Double check that this class works with arrays

Definition at line 228 of file jacobian.h.

## Public Member Functions

- [simple\\_jacobian](#) ()
- virtual int [operator\(\)](#) (size\_t nv, vec\_t &x, vec\_t &y, mat\_t &jac, param\_t &pa)  
*The operator().*

## Data Fields

- double [epsrel](#)  
*The relative stepsize for finite-differencing (default  $10^{-4}$  ).*
- double [epsmin](#)  
*The minimum stepsize (default  $10^{-15}$ ).*
- alloc\_t [ao](#)  
*For memory allocation.*

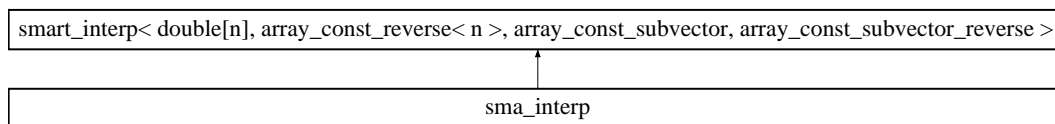
The documentation for this class was generated from the following file:

- jacobian.h

## 3.297 sma\_interp Class Template Reference

```
#include <smart_interp.h>
```

Inheritance diagram for sma\_interp::



### 3.297.1 Detailed Description

```
template<size_t n> class sma_interp< n >
```

A specialization of [smart\\_interp](#) for C-style double arrays.

Definition at line 952 of file smart\_interp.h.

## Public Member Functions

- [sma\\_interp](#) ([base\\_interp](#)< double[n]> &it1, [base\\_interp](#)< [array\\_const\\_reverse](#)< n > > &it2, [base\\_interp](#)< [array\\_const\\_subvector](#)> &it3, [base\\_interp](#)< [array\\_const\\_subvector\\_reverse](#)> &it4)  
*Create with base interpolation objects.*
- [sma\\_interp](#) ()  
*Create with default interpolation objects.*

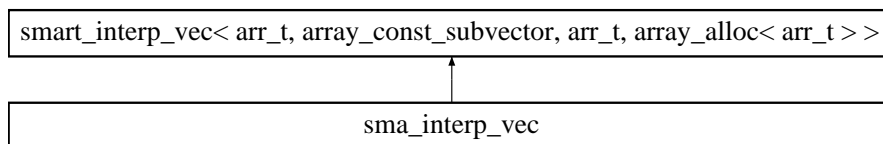
The documentation for this class was generated from the following file:

- smart\_interp.h

## 3.298 sma\_interp\_vec Class Template Reference

```
#include <smart_interp.h>
```

Inheritance diagram for sma\_interp\_vec::



### 3.298.1 Detailed Description

```
template<class arr_t> class sma_interp_vec< arr_t >
```

A specialization of [smart\\_interp\\_vec](#) for C-style arrays.

Definition at line 977 of file smart\_interp.h.

#### Public Member Functions

- [sma\\_interp\\_vec](#) ([base\\_interp](#)< arr\_t > &it, [base\\_interp](#)< [array\\_const\\_subvector](#) > &it2, size\_t n, const arr\_t &x, const arr\_t &y)  
*Create with base interpolation object it and it2.*
- [sma\\_interp\\_vec](#) (size\_t n, const arr\_t &x, const arr\_t &y)  
*Create with default interpolation objects.*

The documentation for this class was generated from the following file:

- smart\_interp.h

## 3.299 smart\_interp Class Template Reference

```
#include <smart_interp.h>
```

### 3.299.1 Detailed Description

```
template<class vec_t = ovector_view, class rvec_t = ovector_const_reverse, class svec_t = ovector_const_subvector, class srvec_t = ovector_const_subvector_reverse> class smart_interp< vec_t, rvec_t, svec_t, srvec_t >
```

Smart interpolation class.

This class can semi-intelligently handle arrays which are not-well formed outside the interpolation region. In particular, if an initial interpolation or derivative calculation fails, the arrays are searched for the largest neighborhood around the point  $x$  for which an interpolation or differentiation will likely produce a finite result.

Definition at line 46 of file smart\_interp.h.

#### Public Member Functions

- [smart\\_interp](#) ([base\\_interp](#)< vec\_t > &it1, [base\\_interp](#)< rvec\_t > &it2, [base\\_interp](#)< svec\_t > &it3, [base\\_interp](#)< srvec\_t > &it4)



- `smart_interp()`
- virtual `~smart_interp()`
- virtual double `interp` (const double x0, size\_t n, const vec\_t &x, const vec\_t &y)  
*Give the value of the function  $y(x = x_0)$ .*
- virtual double `deriv` (const double x0, size\_t n, const vec\_t &x, const vec\_t &y)  
*Give the value of the derivative  $y^{prime}(x = x_0)$ .*
- virtual double `deriv2` (const double x0, size\_t n, const vec\_t &x, const vec\_t &y)  
*Give the value of the second derivative  $y^{prime'}(x = x_0)$ .*
- virtual double `integ` (const double a, const double b, size\_t n, const vec\_t &x, const vec\_t &y)  
*Give the value of the integral  $\int_a^b y(x) dx$ .*

## Data Fields

### Default interpolation objects

- `cspline_interp`< vec\_t > `cit1`
- `cspline_interp`< rvec\_t > `cit2`
- `cspline_interp`< svec\_t > `cit3`
- `cspline_interp`< srvec\_t > `cit4`

## Protected Member Functions

- size\_t `local_lookup` (size\_t n, const vec\_t &x, double x0)  
*A lookup function for generic vectors.*
- int `find_subset` (const double a, const double b, size\_t sz, const vec\_t &x, const vec\_t &y, size\_t &nsz, bool &increasing)  
*Try to find the largest monotonic and finite region around the desired location.*

## Protected Attributes

### Storage internally created subvectors

- const svec\_t \* `sx`
- const svec\_t \* `sy`
- const srvec\_t \* `srx`
- const srvec\_t \* `sry`

### Pointers to interpolation objects

- `base_interp`< vec\_t > \* `rit1`
- `base_interp`< rvec\_t > \* `rit2`
- `base_interp`< svec\_t > \* `rit3`
- `base_interp`< srvec\_t > \* `rit4`

## 3.299.2 Member Function Documentation

**3.299.2.1** `int find_subset` (const double *a*, const double *b*, size\_t *sz*, const vec\_t & *x*, const vec\_t & *y*, size\_t & *nsz*, bool & *increasing*) [`inline`, `protected`]

Try to find the largest monotonic and finite region around the desired location.

### Todo

After row and row2 are set, check to make sure the entire inside region is monotonic before expanding

Definition at line 563 of file `smart_interp.h`.

The documentation for this class was generated from the following file:

- `smart_interp.h`

## 3.300 smart\_interp\_vec Class Template Reference

```
#include <smart_interp.h>
```

### 3.300.1 Detailed Description

```
template<class vec_t, class svec_t, class alloc_vec_t, class alloc_t> class smart_interp_vec< vec_t, svec_t, alloc_vec_t, alloc_t
>
```

Smart interpolation class with pre-specified vectors.

This class can semi-intelligently handle arrays which are not-well formed outside the interpolation region. In particular, if an initial interpolation or derivative calculation fails, the arrays are searched for the largest neighborhood around the point  $x$  for which an interpolation or differentiation will likely produce a finite result.

Definition at line 654 of file smart\_interp.h.

### Public Member Functions

- [smart\\_interp\\_vec](#) (size\_t n, const vec\_t &x, const vec\_t &y)  
*Create with base interpolation objects it and rit.*
- [smart\\_interp\\_vec](#) ([base\\_interp](#)< vec\_t > &it1, [base\\_interp](#)< svec\_t > &it2, size\_t n, const vec\_t &x, const vec\_t &y)  
*Create with base interpolation objects it and rit.*
- virtual [~smart\\_interp\\_vec](#) ()
- virtual double [interp](#) (const double x0)  
*Give the value of the function  $y(x = x_0)$ .*
- virtual double [deriv](#) (const double x0)  
*Give the value of the derivative  $y^{prime}(x = x_0)$ .*
- virtual double [deriv2](#) (const double x0)  
*Give the value of the second derivative  $y^{prime'}(x = x_0)$ .*
- virtual double [integ](#) (const double x1, const double x2)  
*Give the value of the integral  $\int_a^b y(x) dx$ .*

### Data Fields

- [cspline\\_interp](#)< vec\_t > [cit1](#)  
*Default base interpolation object.*
- [cspline\\_interp](#)< svec\_t > [cit2](#)  
*Default base interpolation object.*

### Protected Member Functions

- size\_t [local\\_lookup](#) (size\_t n, const vec\_t &x, double x0)  
*A lookup function for generic vectors.*
- int [find\\_inc\\_subset](#) (const double x0, size\_t sz, const vec\_t &x, const vec\_t &y, size\_t &nusz)  
*Try to find the largest monotonic and finite region around the desired location.*

### Protected Attributes

- svec\_t \* [sx](#)  
*Storage for internally created subvector.*
- svec\_t \* [sy](#)  
*Storage for internally created subvector.*
- [base\\_interp](#)< vec\_t > \* [rit1](#)

- *Pointer to base interpolation object.*
- `base_interp` < svec\_t > \* `rit2`  
*Pointer to base interpolation object.*
- `alloc_t` `ao`  
*Memory allocator for objects of type alloc\_vec\_t.*
- `bool` `inc`  
*True if the user-specified x vector is increasing.*
- `const vec_t` \* `lx`  
*Pointer to user-specified vector.*
- `const vec_t` \* `ly`  
*Pointer to user-specified vector.*
- `alloc_vec_t` `lrx`  
*Reversed version of vector.*
- `alloc_vec_t` `lry`  
*Reversed version of vector.*
- `size_t` `ln`  
*Size of user-specified vector.*

The documentation for this class was generated from the following file:

- smart\_interp.h

## 3.301 sortd Struct Reference

```
#include <table.h>
```

### 3.301.1 Detailed Description

A structure for sorting.

Definition at line 787 of file table.h.

#### Data Fields

- `double` `val`  
*Value to sort.*
- `int` `indx`  
*Sorted index.*

The documentation for this struct was generated from the following file:

- table.h

## 3.302 string\_comp Struct Reference

```
#include <misc.h>
```

### 3.302.1 Detailed Description

Naive string comparison.

This is used internally for the STL routines which require a way to compare strings in the class `table` and in the I/O classes.

Definition at line 125 of file misc.h.

---

## Public Member Functions

- `bool operator()` (const std::string s1, const std::string s2) const  
Return `s1 < s2`.

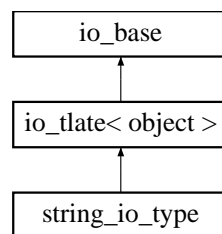
The documentation for this struct was generated from the following file:

- [misc.h](#)

## 3.303 string\_io\_type Class Reference

```
#include <collection.h>
```

Inheritance diagram for string\_io\_type::



### 3.303.1 Detailed Description

I/O object for string variables.

Definition at line 1807 of file collection.h.

## Public Member Functions

- `string_io_type` (const char \*t)  
*Desc.*
- `string_io_type` ()
- int `adds` (collection &co, std::string name, std::string s, bool overwrt=true)  
Add a string to a *collection*.
- std::string `gets` (collection &co, std::string tname)  
Get a string from a *collection*.
- int `get_def` (collection &co, std::string tname, std::string &op, std::string def="")  
Get a string from a *collection*.

The documentation for this class was generated from the following file:

- collection.h

## 3.304 table Class Reference

```
#include <table.h>
```

### 3.304.1 Detailed Description

Data [table](#).

A class to contain and manipulate several equally-sized columns of data.

#### Data representation

Each individual column is just an `ovector_view` (or any descendant of an `ovector_view`) The columns can be referred to in one of two ways:

- A numerical index from 0 to C-1 (where C is the number of columns). For example, data can be accessed through `table::get(size_t c, size_t r)` and `table::set(size_t c, size_t r, double val)`, or the overloaded `[]` operator, `table[c][r]`.
- A name of the column which is a string with no whitespace. For example, data can be accessed with `table::get(string cname, int r)` and `table::set(string cname, int r, double val)`.

The columns are organized in a both a `<map>` and a `<vector>` structure so that finding a column by its index ( `string table::get_column_name(int index)`, and `double table::get_column(int index)` ) takes only constant time, and finding a column by its name ( `int lookup_column()` and `double * table::get_column()` ) is  $O(\log(C))$ . Insertion of a column ( `new_column()` ) is  $O(\log(C))$ , but deletion ( `delete_column()` ) is  $O(C)$ . Adding a row of data can be either  $O(1)$  or  $O(C)$ , but row insertion and deletion is slow, since the all of the columns must be shifted accordingly.

Ownership of any column may be changed at any time, but care must be taken to ensure that memory allocation errors do not occur. These errors should not occur when no columns are owned by the user.

Because of the structure, this class is not suitable for the matrix manipulation. The classes [omatrix](#) and [umatrix](#) are better used for that purpose.

#### Column size

The columns grow automatically (similar to the STL `<vector>`) in response to an attempt to call `set()` for a row that does not presently exist or in a call to `line_of_data()` when the `table` is already full. However, this forces memory rearrangements that are  $O(R \times C)$ . Columns which are not owned by the `table` are not modified, so the `table` will not allow an increase in the number of lines beyond the size of the smallest user-owned column. If the user has a good estimate of the number of rows beforehand, it is best to either specify this in the constructor, or in an explicit call to `inc_maxlines()`.

#### Lookup, differentiation, integration, and interpolation

Lookup, differentiation, integration, and interpolation are automatically implemented using splines from the class `smart_interp_vecp`. A caching mechanism is implemented so that successive interpolations, derivative evaluations or integrations over the same two columns are fast.

#### Sorting

The columns are automatically sorted by name for speed, the results can be accessed by `table::get_sorted_name(i)`. Individual columns can be sorted, or the entire `table` can be sorted by one column.

#### Allowable column names

In general, column names may be of any form as long as they don't contain whitespace, e.g. `123".#$xy~` is a legitimate column name. The column name should be restricted to contain only letters, numbers, and underscores and may not begin with a digit.

#### Thread-safety

Generally, the member functions are thread-safe in the sense that one would expect. Simple `get()` and `set()` functions are thread-safe, while insertion and deletion operations are not. It makes little sense to try to make insertion and deletion thread-safe. The interpolation routines are not thread-safe.

#### I/O and command-line manipulation

When data from an object of type `table` is output to a file through the `collection` class, the `table` can be manipulated on the command-line through the `acol` utility.

## Todo

Move the discussion above to the user guide?

## Todo

Add [interp\(\)](#) and related functions which avoid caching and can thus be const (This has been started with [interp\\_const\(\)](#) )

## Idea for future

The nlines vs. maxlines and automatic resizing of table-owned vs. user-owned vectors could be reconsidered, especially now that ovector can automatically resize on their own.

Definition at line 140 of file table.h.

## Public Member Functions

- [table](#) (int cmaxlines=0)  
*Create a new [table](#) with space for nlines<=cmaxlines.*
- int [set\\_interp](#) ([base\\_interp](#)< [ovector\\_view](#) > &bi1, [base\\_interp](#)< [ovector\\_const\\_subvector](#) > &bi2)  
*Set the base interpolation objects.*
- virtual int [add\\_constant](#) (std::string name, double val)  
*Add a constant.*
- virtual int [remove\\_constant](#) (std::string name)  
*Remove a constant.*

## Basic get and set methods

- int [set](#) (std::string col, size\_t row, double val)  
*Set row row of column named col to value val -  $O(\log(C))$ .*
- int [set](#) (size\_t icol, size\_t row, double val)  
*Set row row of column number icol to value val -  $O(1)$ .*
- double [get](#) (std::string col, size\_t row) const  
*Get value from row row of column named col -  $O(\log(C))$ .*
- double [get](#) (size\_t icol, size\_t row)  
*Get value from row row of column number icol -  $O(1)$ .*
- int [get\\_ncolumns](#) () const  
*Return the number of columns.*
- size\_t [get\\_nlines](#) () const  
*Return the number of lines.*
- int [set\\_nlines](#) (size\_t il)  
*Set the number of lines.*
- int [set\\_nlines\\_auto](#) (size\_t il)  
*Set the number of lines.*
- int [get\\_maxlines](#) ()  
*Return the maximum number of lines.*
- [ovector\\_view](#) \* [get\\_column](#) (std::string col)  
*Returns a pointer to the column named col -  $O(\log(C))$ .*
- const [ovector\\_view](#) \* [get\\_column\\_const](#) (std::string col) const  
*Returns a pointer to the column named col -  $O(\log(C))$ .*
- [ovector\\_view](#) \* [get\\_column](#) (size\_t icol)  
*Returns a pointer to the column of index icol -  $O(1)$ .*
- const [ovector\\_view](#) \* [get\\_column](#) (size\_t icol) const  
*Returns a pointer to the column of index icol -  $O(1)$ .*
- const [ovector\\_view](#) & [operator\[\]](#) (size\_t icol) const  
*Returns the column of index icol -  $O(1)$  (const version).*
- [ovector\\_view](#) & [operator\[\]](#) (size\_t icol)  
*Returns the column of index icol -  $O(1)$ .*
- const [ovector\\_view](#) & [operator\[\]](#) (std::string scol) const  
*Returns the column named scol -  $O(\log(C))$  (const version).*

- `ovector_view` & `operator[]` (std::string scol)  
*Returns the column named scol -  $O(\log(C))$ .*
- int `get_row` (std::string col, double val, `ovector` &row) const  
*Returns a pointer to a copy of the row with value val in column col -  $O(R*C)$ .*
- int `get_row` (size\_t irow, `ovector` &row) const  
*Returns a pointer to a copy of row number irow -  $O(C)$ .*

### Column manipulation

- std::string `get_column_name` (size\_t col) const  
*Returns the name of column col -  $O(1)$ .*
- std::string `get_sorted_name` (size\_t col)  
*Returns the name of column col in sorted order -  $O(1)$ .*
- int `new_column` (std::string name)  
*Add a new column owned by the table -  $O(\log(C))$ .*
- int `new_column` (std::string name, `ovector_view` \*ldat)  
*Add a new column owned by the user -  $O(\log(C))$ .*
- int `lookup_column` (std::string name, int &ix)  
*Find the index for column named name -  $O(\log(C))$ .*
- int `rename_column` (std::string olds, std::string news)  
*Rename column named olds to news -  $O(C)$ .*
- int `copy_column` (std::string src, std::string dest)  
*Make a new column named dest equal to src -  $O(\log(C)*R)$ .*
- double \* `create_array` (std::string col) const  
*Create (using new) a generic array from column col.*
- int `init_column` (std::string scol, double val)  
*Initialize all values of column named scol to val -  $O(\log(C)*R)$ .*
- int `ch_owner` (std::string name, bool ow)  
*Modify ownership -  $O(\log(C))$ .*
- bool `get_owner` (std::string name) const  
*Get ownership -  $O(\log(C))$ .*
- const gsl\_vector \* `get_gsl_vector` (std::string name) const  
*Get a gsl\_vector from column name -  $O(\log(C))$ .*
- int `check_synchro` () const  
*Return 0 if the tree and list are properly synchronized.*
- int `add_col_from_table` (std::string loc\_index, `table` &source, std::string src\_index, std::string src\_col, std::string dest\_col="")  
*Insert a column from a separate table, interpolating it into a new column.*

### Row manipulation and data input

- int `new_row` (size\_t n)  
*Insert a row before row n.*
- int `copy_row` (size\_t src, size\_t dest)  
*Copy the data in row src to row dest.*
- int `insert_data` (size\_t n, size\_t nv, double \*v)  
*Insert a row of data before row n.*
- int `insert_data` (size\_t n, size\_t nv, double \*\*v)  
*Insert a row of data before row n.*
- int `line_of_names` (std::string newheads)  
*Read a new set of names from newheads.*
- template<class vec\_t>  
int `line_of_data` (size\_t nv, const vec\_t &v)  
*Read a line of data from an array.*

### Lookup and search methods

- size\_t `ordered_lookup` (std::string col, double val)  
*Look for a value in an ordered column.*
- size\_t `lookup` (std::string col, double val) const  
*Exhaustively search column col for the value val -  $O(\log(C)*R)$ .*

- double `lookup_val` (std::string col, double val, std::string col2) const  
*Search column col for the value val and return value in col2.*
- size\_t `lookup` (int col, double val) const  
*Exhaustively search column col for the value val -  $O(\log(C)*R)$ .*
- size\_t `mlookup` (std::string col, double val, std::vector< double > &results, double threshold=0.0) const  
*Exhaustively search column col for many occurrences of val -  $O(\log(C)*R)$ .*
- int `lookup_form` (std::string formula, double &maxval)  
*Search for row with maximum value of formula.*

### Interpolation, differentiation, and integration, max, and min

- double `interp` (std::string sx, double x0, std::string sy)  
*Interpolate x0 from sx into sy.*
- double `interp_const` (std::string sx, double x0, std::string sy) const  
*Interpolate x0 from sx into sy.*
- double `interp` (size\_t ix, double x0, size\_t iy)  
*Interpolate x0 from ix into iy.*
- int `deriv` (std::string x, std::string y, std::string yp)  
*Make a new column yp which is the derivative  $y'(x)$  -  $O(\log(C)*R)$ .*
- double `deriv` (std::string sx, double x0, std::string sy)  
*The first derivative of the function sy(sx) at  $sx=x0$ .*
- double `deriv` (size\_t ix, double x0, size\_t iy)  
*The first derivative of the function iy(ix) at  $ix=x0$ .*
- int `deriv2` (std::string x, std::string y, std::string yp)  
*Make a new column yp which is  $y''(x)$  -  $O(\log(C)*R)$ .*
- double `deriv2` (std::string sx, double x0, std::string sy)  
*The second derivative of the function sy(sx) at  $sx=x0$ .*
- double `deriv2` (size\_t ix, double x0, size\_t iy)  
*The second derivative of the function iy(ix) at  $ix=x0$ .*
- double `integ` (std::string sx, double x1, double x2, std::string sy)  
*The integral of the function sy(sx) from  $sx=x1$  to  $sx=x2$ .*
- double `integ` (size\_t ix, double x1, double x2, size\_t iy)  
*The integral of the function iy(ix) from  $ix=x1$  to  $ix=x2$ .*
- int `integ` (std::string x, std::string y, std::string ynew)  
*The integral of the function iy(ix).*
- double `max` (std::string col) const  
*Return column maximum. Makes no assumptions about ordering -  $O(R)$ .*
- double `min` (std::string col) const  
*Return column minimum. Makes no assumptions about ordering -  $O(R)$ .*

### Subtable method

- `table * subtable` (std::string list, size\_t top, size\_t bottom, bool linked=true)  
*Make a subtable.*

### Add space

- int `inc_maxlines` (size\_t llines)  
*Manually increase the maximum number of lines.*

### Delete methods

- int `delete_column` (std::string scol)  
*Delete column named scol -  $O(C)$ .*
- int `delete_row` (std::string scol, double val)  
*Delete the row with the value val in column scol.*
- int `delete_row` (size\_t irow)  
*Delete the row of index irow.*

### Clear methods



- void [zero\\_table](#) ()  
*Zero the data entries but keep the column names and nlines fixed.*
- void [clear\\_table](#) ()  
*Clear the [table](#) and the column names.*
- void [clear\\_data](#) ()  
*Remove all of the data by setting the number of lines to zero.*

### Sorting methods

- int [sort\\_table](#) (std::string scol)  
*Sort the entire [table](#) by the column `scol`.*
- int [sort\\_column](#) (std::string scol)  
*Individually sort the column `scol`.*

### Summary method

- int [summary](#) (std::ostream \*out, int ncol=79) const  
*Output a summary of the information stored.*

### Data Fields

- std::map< std::string, double > [constants](#)  
*The list of constants.*

### Protected Types

#### Iterator types

- typedef std::map< std::string, [col](#), [string\\_comp](#) >::iterator [aiter](#)
- typedef std::map< std::string, [col](#), [string\\_comp](#) >::const\_iterator [aciter](#)
- typedef std::vector< [aiter](#) >::iterator [aviter](#)

### Protected Member Functions

- int [reset\\_list](#) ()  
*Set the elements of alist with the appropriate iterators from atree -  $O(C)$ .*

### Column manipulation methods

- [aiter get\\_iterator](#) (std::string lname)  
*Return the iterator for a column.*
- [col](#) \* [get\\_col\\_struct](#) (std::string lname)  
*Return the column structure for a column.*
- [aiter begin](#) ()  
*Return the beginning of the column tree.*
- [aiter end](#) ()  
*Return the end of the column tree.*

### Static Protected Member Functions

- static int [sortd\\_comp](#) (const void \*a, const void \*b)  
*The sorting function.*

## Protected Attributes

### Actual data

- `size_t maxlines`  
*The size of allocated memory.*
- `size_t nlines`  
*The size of presently used memory.*
- `std::map< std::string, col, string_comp > atree`  
*The tree of columns.*
- `std::vector< aiter > alist`  
*The list of tree iterators.*

### Interpolation

- `sm_interp_vec * si`  
*The interpolation object.*
- `base_interp< ovector_view > * intp1`  
*A pointer to the interpolation object.*
- `base_interp< ovector_const_subvector > * intp2`  
*A pointer to the subvector interpolation object.*
- `cspline_interp< ovector_view > cintp1`  
*The default interpolation object.*
- `cspline_interp< ovector_const_subvector > cintp2`  
*The default subvector interpolation object.*
- `search_vec< vec_t > se`  
*The vector-searching object.*
- `int itype`  
*The interpolation type.*
- `bool intp_set`  
*True if the interpolation type has been set.*
- `std::string intp_colx`  
*The last x-column interpolated.*
- `std::string intp_coly`  
*The last y-column interpolated.*

## 3.304.2 Member Function Documentation

### 3.304.2.1 `int set (std::string col, size_t row, double val)`

Set row `row` of column named `col` to value `val` -  $O(\log(C))$ .

This function adds the column `col` if it does not already exist and adds rows using `inc_maxlines()` and `set_nlines()` to create at least  $(row+1)$  rows if they do not already exist.

### 3.304.2.2 `int set_nlines (size_t il)`

Set the number of lines.

This function is stingy about increasing the `table` memory space and will only increase it enough to fit `il` lines, which is useful if you have columns not owned by the `table`.

### 3.304.2.3 `int set_nlines_auto (size_t il)`

Set the number of lines.

## Todo

Resolve whether `set()` should really use this approach. Also, resolve whether this should replace `set_nlines()` (It could be that the answer is no, because as the documentation in the other version states, the other version is useful if you have columns not owned by the `table`.)

**3.304.2.4** `ovector_view* get_column (size_t icol) [inline]`

Returns a pointer to the column of index `icol` -  $O(1)$ .

Note that several of the methods require reallocation of memory and pointers previously returned by this function will be incorrect.

Definition at line 221 of file `table.h`.

**3.304.2.5** `const ovector_view* get_column (size_t icol) const [inline]`

Returns a pointer to the column of index `icol` -  $O(1)$ .

Note that several of the methods require reallocation of memory and pointers previously returned by this function will be incorrect.

Definition at line 232 of file `table.h`.

**3.304.2.6** `const ovector_view& operator[ ] (size_t icol) const [inline]`

Returns the column of index `icol` -  $O(1)$  (const version).

This does not do any sort of bounds checking and is quite fast.

Note that several of the methods require reallocation of memory and references previously returned by this function will be incorrect.

Definition at line 246 of file `table.h`.

**3.304.2.7** `ovector_view& operator[ ] (size_t icol) [inline]`

Returns the column of index `icol` -  $O(1)$ .

This does not do any sort of bounds checking and is quite fast.

Note that several of the methods require reallocation of memory and references previously returned by this function will be incorrect.

Definition at line 260 of file `table.h`.

**3.304.2.8** `const ovector_view& operator[ ] (std::string scol) const [inline]`

Returns the column named `scol` -  $O(\log(C))$  (const version).

No error checking is performed.

Note that several of the methods require reallocation of memory and references previously returned by this function will be incorrect.

Definition at line 273 of file `table.h`.

**3.304.2.9** `ovector_view& operator[ ] (std::string scol) [inline]`

Returns the column named `scol` -  $O(\log(C))$ .

No error checking is performed.

Note that several of the methods require reallocation of memory and references previously returned by this function will be incorrect.

Definition at line 287 of file `table.h`.

**3.304.2.10** `int new_column (std::string name, ovector_view * ldat)`

Add a new column owned by the user -  $O(\log(C))$ .

This function does not modify the number of lines of data in the [table](#).

**Todo**

We've got to figure out what to do if `ldat` is too small. If it's smaller than `nlines`, obviously we should just fail, but what if it's size is between `nlines` and `maxlines`?

**3.304.2.11 int lookup\_column (std::string name, int & ix)**

Find the index for column named `name` -  $O(\log(C))$ .

If the column is not present, this does not call the error handler, but quietly sets `ix` to zero and returns `gsl_notfound`.

**3.304.2.12 int rename\_column (std::string olds, std::string news)**

Rename column named `olds` to `news` -  $O(C)$ .

This is slow since we have to delete the column and re-insert it. This process in turn mangles all of the iterators in the list.

**3.304.2.13 int init\_column (std::string scol, double val)**

Initialize all values of column named `scol` to `val` -  $O(\log(C)*R)$ .

Note that this does not initialize elements beyond `nlines` so that if the number of rows is increased afterwards, the new rows will have uninitialized values.

**3.304.2.14 int ch\_owner (std::string name, bool ow)**

Modify ownership -  $O(\log(C))$ .

Warning: columns allocated using `malloc()` should never be owned by the `table` object since it uses `delete` instead of `free()`.

**3.304.2.15 int add\_col\_from\_table (std::string loc\_index, table & source, std::string src\_index, std::string src\_col, std::string dest\_col = "")**

Insert a column from a separate `table`, interpolating it into a new column.

Given a pair of columns (`src_index`, `src_col`) in a separate `table` (`source`), this creates a new column in the present `table` named `src_col` which interpolates `loc_index` into `src_index`. The interpolation type from the `source table` will be used and the interpolation type of the present `table` is ignored. If there is already a column in the present `table` named `src_col`, then this will fail.

If there is an error in the interpolation for any particular row, then the value of `src_col` in that row will be set to zero.

**3.304.2.16 size\_t ordered\_lookup (std::string col, double val)**

Look for a value in an ordered column.

$O(\log(C)*\log(R))$

**3.304.2.17 int lookup\_form (std::string formula, double & maxval)**

Search for row with maximum value of formula.

This searches the `table` for the maximum value of the specified formula. This is useful in several ways. For example, to find the row for which the column `mu` is 2 and `T` is 3, you can call

```
table::lookup_form("-abs(mu-2)-abs(T-3)");
```

**3.304.2.18 double interp (std::string sx, double x0, std::string sy)**

Interpolate  $x_0$  from  $s_x$  into  $s_y$ .

$O(\log(C) \cdot \log(R))$  but can be as bad as  $O(\log(C) \cdot R)$  if the relevant columns are not well ordered.

**3.304.2.19 double interp\_const (std::string sx, double x0, std::string sy) const**

Interpolate  $x_0$  from  $s_x$  into  $s_y$ .

$O(\log(C) \cdot \log(R))$  but can be as bad as  $O(\log(C) \cdot R)$  if the relevant columns are not well ordered.

**3.304.2.20 double interp (size\_t ix, double x0, size\_t iy)**

Interpolate  $x_0$  from  $i_x$  into  $i_y$ .

$O(\log(R))$  but can be as bad as  $O(R)$  if the relevant columns are not well ordered.

**3.304.2.21 double deriv (std::string sx, double x0, std::string sy)**

The first derivative of the function  $s_y(s_x)$  at  $s_x=x_0$ .

$O(\log(C) \cdot \log(R))$  but can be as bad as  $O(\log(C) \cdot R)$  if the relevant columns are not well ordered.

**3.304.2.22 double deriv (size\_t ix, double x0, size\_t iy)**

The first derivative of the function  $i_y(i_x)$  at  $i_x=x_0$ .

$O(\log(R))$  but can be as bad as  $O(R)$  if the relevant columns are not well ordered.

**3.304.2.23 double deriv2 (std::string sx, double x0, std::string sy)**

The second derivative of the function  $s_y(s_x)$  at  $s_x=x_0$ .

$O(\log(C) \cdot \log(R))$  but can be as bad as  $O(\log(C) \cdot R)$  if the relevant columns are not well ordered.

**3.304.2.24 double deriv2 (size\_t ix, double x0, size\_t iy)**

The second derivative of the function  $i_y(i_x)$  at  $i_x=x_0$ .

$O(\log(R))$  but can be as bad as  $O(R)$  if the relevant columns are not well ordered.

**3.304.2.25 double integ (std::string sx, double x1, double x2, std::string sy)**

The integral of the function  $s_y(s_x)$  from  $s_x=x_1$  to  $s_x=x_2$ .

$O(\log(C) \cdot \log(R))$  but can be as bad as  $O(\log(C) \cdot R)$  if the relevant columns are not well ordered.

**3.304.2.26 double integ (size\_t ix, double x1, double x2, size\_t iy)**

The integral of the function  $i_y(i_x)$  from  $i_x=x_1$  to  $i_x=x_2$ .

$O(\log(R))$  but can be as bad as  $O(R)$  if the relevant columns are not well ordered.

**3.304.2.27 int integ (std::string x, std::string y, std::string ynew)**

The integral of the function  $i_y(i_x)$ .

$O(\log(R))$  but can be as bad as  $O(R)$  if the relevant columns are not well ordered.

**3.304.2.28 table\* subtable (std::string list, size\_t top, size\_t bottom, bool linked = true)**

Make a subtable.

Uses the columns specified in `list` from the row `top` to the row of index `bottom`. If `linked` is false the the data will be independent from the original [table](#).

**3.304.2.29 int delete\_column (std::string scol)**

Delete column named `scol` - O(C).

This is slow because the iterators in `alist` are mangled and we have to call `reset_list` to get them back.

**3.304.2.30 void clear\_data () [inline]**

Remove all of the data by setting the number of lines to zero.

This leaves the column names intact and does not remove the constants.

Definition at line 669 of file `table.h`.

**3.304.2.31 int summary (std::ostream \* out, int ncol = 79) const**

Output a summary of the information stored.

Outputs the number of constants, the number of columns, a list of the column names, and the number of lines of data.

**3.304.2.32 int reset\_list () [protected]**

Set the elements of `alist` with the appropriate iterators from `atree` - O(C).

Generally, the end-user shouldn't need this method. It is only used in [delete\\_column\(\)](#) to rearrange the list when a column is deleted from the tree.

The documentation for this class was generated from the following file:

- `table.h`

**3.305 test\_mgr Class Reference**

```
#include <test_mgr.h>
```

**3.305.1 Detailed Description**

A class to manage testing and record success and failure.

**Idea for future**

[test\\_mgr::success](#) and [test\\_mgr::last\\_fail](#) should be protected, but that breaks the [operator+\(\)](#) function. Can this be fixed?

Definition at line 38 of file `test_mgr.h`.

**Public Member Functions**

- [test\\_mgr \(\)](#)
- [bool report \(\)](#)

*Provide a report of all tests so far.*

- `std::string get_last_fail ()`  
*Returns the description of the last test that failed.*
- `void set_output_level (int l)`  
*Set the output level.*
- `int get_ntests ()`  
*Return the number of tests performed so far.*

### The testing methods

- `bool test_rel (double result, double expected, double rel_error, std::string description)`  
*Test for  $|result - expected|/expected < rel\_error$ .*
- `bool test_abs (double result, double expected, double abs_error, std::string description)`  
*Test for  $|result - expected|/ < abs\_error$ .*
- `bool test_fact (double result, double expected, double factor, std::string description)`  
*Test for  $1/factor < result/expected < factor$  ??*
- `bool test_str (std::string result, std::string expected, std::string description)`  
*Test for result = expected.*
- `bool test_gen (bool value, std::string description)`  
*Test for result = expected.*
- `template<class vec_t>`  
`bool test_rel_arr (int nv, vec_t &result, vec_t &expected, double rel_error, std::string description)`  
*Test for  $|result - expected|/expected < rel\_error$ .*
- `template<class vec_t>`  
`bool test_abs_arr (int nv, vec_t &result, vec_t &expected, double rel_error, std::string description)`  
*Test for  $|result - expected|/ < abs\_error$ .*
- `template<class vec_t>`  
`bool test_fact_arr (int nv, vec_t &result, vec_t &expected, double factor, std::string description)`  
*Test for  $1/factor < result/expected < factor$  ??*
- `bool test_int_arr (int nv, int *result, int *expected, std::string description)`  
*Test for result = expected.*

### Data Fields

- `bool success`  
*True if all tests have passed.*
- `std::string last_fail`  
*The description of the last failed test.*

### Protected Member Functions

- `void process_test (bool ret, std::string d2, std::string description)`  
*A helper function for processing tests.*

### Protected Attributes

- `int ntests`  
*The number of tests performed.*
- `int output_level`  
*The output level.*

### Friends

- `const test_mgr operator+ (const test_mgr &left, const test_mgr &right)`  
*Add two test\_mgr objects (if either failed, the sum fails).*

### 3.305.2 Member Function Documentation

#### 3.305.2.1 bool report ()

Provide a report of all tests so far.

Returns true if all tests have passed and false if at least one test failed.

#### 3.305.2.2 void set\_output\_level (int *l*) [inline]

Set the output level.

Possible values:

- 0 = No output
- 1 = Output only tests that fail
- 2 = Output all tests

Definition at line 78 of file test\_mgr.h.

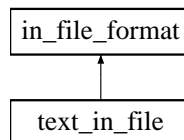
The documentation for this class was generated from the following file:

- test\_mgr.h

## 3.306 text\_in\_file Class Reference

```
#include <text_file.h>
```

Inheritance diagram for text\_in\_file::



### 3.306.1 Detailed Description

An input text file.

**Note:** Text files are not entirely architecture-independent, For example, a larger integer will not be read correctly on small integer systems.

Definition at line 242 of file text\_file.h.

#### Public Member Functions

- [text\\_in\\_file](#) (std::istream \*in\_file)  
*Use input stream in\_file for text input.*
- [text\\_in\\_file](#) (std::string file\_name)  
*Read an input file with name file\_name.*
- virtual int [bool\\_in](#) (bool &dat, std::string name="")  
*Input a bool variable.*
- virtual int [char\\_in](#) (char &dat, std::string name="")  
*Input a char variable.*



- virtual int `double_in` (double &dat, std::string name="")  
*Input a double variable.*
- virtual int `float_in` (float &dat, std::string name="")  
*Input a float variable.*
- virtual int `int_in` (int &dat, std::string name="")  
*Input an int variable.*
- virtual int `long_in` (unsigned long int &dat, std::string name="")  
*Input an long variable.*
- virtual int `string_in` (std::string &dat, std::string name="")  
*Input a string variable.*
- virtual int `word_in` (std::string &dat, std::string name="")  
*Input a word variable.*
- virtual int `start_object` (std::string &type, std::string &name)  
*Start object input.*
- virtual int `skip_object` ()  
*Skip the present object for the next call to read\_type().*
- virtual int `end_object` ()  
*End object input.*
- virtual int `init_file` ()  
*Initialize file input.*
- virtual int `clean_up` ()  
*Finish file input.*
- std::string `reformat_string` (std::string in)  
*Add brackets and replace carriage returns with spaces.*

### Protected Member Functions

- bool `is_hc_type` (std::string type)  
*If true, then type is a "hard-coded" type.*
- virtual int `word_in_noerr` (std::string &dat, std::string name="")  
*A version of word\_in() which doesn't call the error handler.*

### Protected Attributes

- std::stack< bool > `hcs`  
*A list to indicate if the current object and subobjects are "hard-coded".*
- std::istream \* `ins`  
*The input stream.*
- bool `from_string`  
*True if the string version of the constructor was called.*

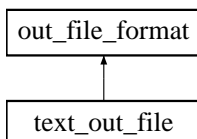
The documentation for this class was generated from the following file:

- text\_file.h

## 3.307 text\_out\_file Class Reference

```
#include <text_file.h>
```

Inheritance diagram for text\_out\_file::



### 3.307.1 Detailed Description

An output text file.

A [collection](#) file is simply a text file containing a list of objects specially formatted for input and output. Each entry in the text file is of the form:

```
object_type object_name object_version word_number word1 word2 word3 ...
```

The type, name, and version are all strings that contain no whitespace. "a\_name" is a valid name but, "a name" is not.

Parts of the object definition may be separated by any amount of whitespace, with the exception of 'strings'.

The [collection](#) file may contain comments, which are lines that begin with the '#' character. Comments may last more than one line (so long as every line begins with '#'), but they may not occur in the middle of an object definition.

```
# comment 1
double a 4.0
double[] c 4
4.5 3.4 2.3 4.2
# comment 2
double b 5.0
```

is acceptable, but

```
# comment 1
double a 4.0
double[] c 4
4.5 3.4 2.3 4.2
double b
# comment 2
5.0
```

is not, since the second comment occurs in the middle of the definition for the object named 'b'.

Normal variable: type name version word data1 ... data2

Object containing pointer: type name version word data1 ... ptr\_type ptr\_name ... data2

where ptr\_type is the type of the object being pointed to and ptr\_name is the name of the object. If it's not in the list, then the object is assigned a unique name of the form ptrX where 'X' is an integer  $\geq 0$ .

Static objects are the same, except they are preceeded by the keyword `static` and do not have a name associated with them.

This is useful for output to text files and to `std::cout`, i.e. [text\\_out\\_file](#) tf(&cout);

**Note:** Text files are not entirely architecture-independent, For example, a larger integer will not be read correctly on small integer systems.

Definition at line 97 of file `text_file.h`.

### Public Member Functions

- [text\\_out\\_file](#) (std::ostream \*out\_file, int width=80)  
*Use output stream out\_file for text output.*
- [text\\_out\\_file](#) (std::string file\_name, std::ostream \*prop=NULL, bool append=false, int width=80)  
*Create an output file with name file\_name.*
- virtual int [bool\\_out](#) (bool dat, std::string name="")  
*Output a bool variable.*
- virtual int [char\\_out](#) (char dat, std::string name="")  
*Output a char variable.*
- virtual int [char\\_out\\_internal](#) (char dat, std::string name="")

- *Output a char variable.*
- virtual int `double_out` (double dat, std::string name="")  
*Output a double variable.*
- virtual int `float_out` (float dat, std::string name="")  
*Output a float variable.*
- virtual int `int_out` (int dat, std::string name="")  
*Output an int variable.*
- virtual int `long_out` (unsigned long int dat, std::string name="")  
*Output an long variable.*
- virtual int `string_out` (std::string dat, std::string name="")  
*Output a string.*
- virtual int `word_out` (std::string dat, std::string name="")  
*Output a word.*
- virtual int `start_object` (std::string type, std::string name)  
*Start object output.*
- virtual int `end_object` ()  
*End object output.*
- virtual int `end_line` ()  
*End line.*
- virtual int `init_file` ()  
*Output initialization.*
- virtual int `clean_up` ()  
*Finish the file.*
- int `comment_out` (std::string comment)  
*Output a comment (only for text files).*
- std::string `reformat_string` (std::string in)  
*Add brackets and replace carriage returns with spaces.*

### Protected Member Functions

- virtual int `flush` ()  
*Flush the string buffer.*
- bool `is_hc_type` (std::string type)  
*If true, then type is a "hard-coded" type.*

### Protected Attributes

- std::stack< bool > `hcs`  
*A list to indicate if the current object and subobjects are "hard-coded".*
- bool `from_string`  
*True if the constructor was called with a string, false otherwise.*
- bool `compressed`  
*True if the file is to be compressed.*
- bool `gzip`  
*True if the file is to be compressed with gzip.*
- int `file_width`  
*The width of the file.*
- std::ostream \* `outs`  
*The output stream.*
- std::ostream \* `props`  
*A pointer to an output stream to define output properties.*
- std::ostringstream \* `strout`  
*The temporary buffer as a stringstream.*
- std::string `user_filename`  
*The user-specified filename.*
- std::string `temp_filename`  
*The temporary filename used.*

### 3.307.2 Constructor & Destructor Documentation

#### 3.307.2.1 text\_out\_file (std::ostream \* out\_file, int width = 80)

Use output stream `out_file` for text output.

This constructor assumes that the I/O properties of `out_file` have already been set.

Note that the stream `out_file` should not have been opened in binary mode, and errors will likely occur if this is the case.

#### Todo

Ensure streams are not opened in binary mode for safety.

#### 3.307.2.2 text\_out\_file (std::string file\_name, std::ostream \* prop = NULL, bool append = false, int width = 80)

Create an output file with name `file_name`.

If `prop` is not null, then the I/O properties (precision, fill, flags, etc) for the newly created file are taken to be the same as `prop`.

The documentation for this class was generated from the following file:

- `text_file.h`

## 3.308 timer\_clock Class Reference

```
#include <timer.h>
```

### 3.308.1 Detailed Description

Provide an interface for timing execution using `clock()`.

#### Note:

Note that the time return by `clock()` is reset on some regular interval (sometimes 72 minutes) and this class does not yet account for this.

Definition at line 116 of file `timer.h`.

#### Public Member Functions

- double [time\\_since](#) ()  
*Number of seconds elapsed.*
- void [time\\_since](#) (int &d, int &h, int &m, int &s, double &f)  
*Time elapsed in days, hours, minutes, seconds, and fractions of seconds.*
- void [time\\_remaining](#) (int n, int tot, int &d, int &h, int &m, int &s, double &f)  
*Time remaining if n out of tot tasks have been completed.*
- std::string [interval\\_to\\_string](#) (int d, int h, int m, int s, double f=0.0)  
*Convert a time interval to a string.*

#### Protected Attributes

- clock\_t [time](#)  
*Desc.*

The documentation for this class was generated from the following file:

- `timer.h`

## 3.309 timer\_gettod Class Reference

```
#include <timer.h>
```

### 3.309.1 Detailed Description

Provide an interface for timing execution using `gettimeofday()`.

#### Todo

Better testing which doesn't use a fixed number of mathematical operations, but automatically selects enough operations.

Definition at line 44 of file `timer.h`.

### Public Member Functions

- [timer\\_gettod\(\)](#)
- [int reset\(\)](#)  
*Set time 'zero'.*
- [int set\(\)](#)  
*Store the present time.*
- [double seconds\\_elapsed\(\)](#)  
*Return the number of seconds between [set\(\)](#) and [reset\(\)](#).*
- [int time\\_elapsed\(int &d, int &h, int &m, int &s, int &usec\)](#)  
*Return the time between [set\(\)](#) and [reset\(\)](#).*
- [int time\\_remaining\(int n, int ntot, int &d, int &h, int &m, int &s, int &usec\)](#)  
*Time remaining if  $n$  out of  $ntot$  tasks have been completed.*
- [std::string time\\_remaining\(int n, int ntot\)](#)  
*Time remaining if  $n$  out of  $ntot$  tasks have been completed.*
- [std::string interval\\_to\\_string\(int d, int h, int m, int s, int usec\)](#)  
*Convert a time interval to a string.*

### Protected Attributes

- [struct timeval zero](#)  
*The last time the clock was reset.*
- [struct timeval mark](#)  
*The most resent time from [set\(\)](#).*
- [struct timezone tz](#)  
*The timezone.*

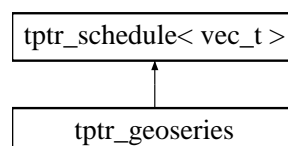
The documentation for this class was generated from the following file:

- `timer.h`

## 3.310 tptr\_geoseries Class Template Reference

```
#include <tptr_geoseries.h>
```

Inheritance diagram for `tptr_geoseries`:



### 3.310.1 Detailed Description

**template<class vec\_t = ovector\_view> class tptr\_geoseries< vec\_t >**

Temperature schedule for a geometric series.

The temperature begins at `start`, and is divided by `ratio`, until it is smaller than `end`. The ending value is divided by `sqrt(ratio)` to avoid finite precision problems for series when `start/end` is an integral power of `ratio`.

The default schedule is  $T = 1/(1.01)^n$  for  $n = 0, 1, 2, 3, \dots, 463$  (until  $T < 0.01$ ) given by `ustart=1`, `uend=0.01`, `uratio=1.01`.

Definition at line 49 of file `tptr_geoseries.h`.

#### Public Member Functions

- [tptr\\_geoseries](#) ()
- virtual [~tptr\\_geoseries](#) ()
- int [set\\_series](#) (double ustart, double uend, double uratio)  
*Set the limits for the geometric series.*
- int [get\\_npoints](#) ()  
*Get the number of temperatures in the series.*
- virtual double [start](#) (double min, int nv, const vec\_t &best, void \*vp)  
*Return the first temperature.*
- virtual double [next](#) (double min, int nv, const vec\_t &best, void \*vp)  
*Return the next temperature.*
- virtual bool [done](#) (double min, int nv, const vec\_t &best, void \*vp)  
*Return true if the last step made the temperature too small.*
- virtual const char \* [type](#) ()  
*Return string denoting type ("tptr\_geoseries").*

#### Protected Attributes

- double [last](#)  
*The last temperature returned.*

#### parameters for the schedule

- double [dstart](#)
- double [end](#)
- double [ratio](#)

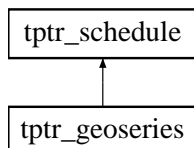
The documentation for this class was generated from the following file:

- `tptr_geoseries.h`

## 3.311 tptr\_schedule Class Template Reference

```
#include <tptr_schedule.h>
```

Inheritance diagram for `tptr_schedule`:



### 3.311.1 Detailed Description

**template<class vec\_t = ovector\_view> class tptr\_schedule< vec\_t >**

Simulated annealing temperature schedule base.

The schedules are designed to be used in the following way

```
for (temper=tp->start (fmin,nvar,pb,NULL) ;
    tp->done (fmin,nvar,pb,NULL) ==false;
    temper=tp->next (fmin,nvar,pb,NULL) ) {
}
```

Definition at line 48 of file tptr\_schedule.h.

### Public Member Functions

- virtual `~tptr_schedule ()`
- virtual double `start` (double min, int nv, const vec\_t &best, void \*vp)  
*Return the first temperature.*
- virtual double `next` (double min, int nv, const vec\_t &best, void \*vp)  
*Return the next temperature.*
- virtual bool `done` (double min, int nv, const vec\_t &best, void \*vp)  
*Return true if the last step made the temperature too small.*
- virtual const char \* `type` ()  
*Return string denoting type ("tptr\_schedule").*

The documentation for this class was generated from the following file:

- tptr\_schedule.h

## 3.312 twod\_eqi\_intp Class Reference

```
#include <twod_eqi_intp.h>
```

### 3.312.1 Detailed Description

Two-dimensional interpolation for equally-spaced intervals.

This implements the relations from Abramowitz and Stegun:

$$f(x_0 + ph, y_0 + qk) =$$

3-point

$$(1 - p - q)f_{0,0} + pf_{1,0} + qf_{0,1}$$

4-point

$$(1 - p)(1 - q)f_{0,0} + p(1 - q)f_{1,0} + q(1 - p)f_{0,1} + pqf_{1,1}$$

6-point

$$\frac{q(q-1)}{2}f_{0,-1} + \frac{p(p-1)}{2}f_{-1,0} + (1 + pq - p^2 - q^2)f_{0,0} + \frac{p(p-2q+1)}{2}f_{1,0} + \frac{q(q-2p+1)}{2}f_{0,1} + pqf_{1,1}$$

Definition at line 55 of file twod\_eqi\_intp.h.

## Public Member Functions

- double [interp](#) (double x, double y)  
*Perform the 2-d interpolation.*
- int [set\\_type](#) (int type)  
*Set the interpolation type.*

## Data Fields

- double [xoff](#)  
*Offset in x-direction.*
- double [yoff](#)  
*Offset in y-direction.*

### 3.312.2 Member Function Documentation

#### 3.312.2.1 int set\_type (int type) [inline]

Set the interpolation type.

- 3: 3-point
- 4: 4-point
- 6: 6-point (default)

Definition at line 78 of file twod\_eqi\_intp.h.

The documentation for this class was generated from the following file:

- twod\_eqi\_intp.h

## 3.313 twod\_intp Class Reference

```
#include <twod_intp.h>
```

### 3.313.1 Detailed Description

Two-dimensional interpolation class.

This class implements two-dimensional interpolation. Derivatives and integrals along both x- and y-directions can be computed.

The storage for the data, including the arrays `x_fun` and `y_fun` are all managed by the user. If the data is changed without calling [reset\\_interp\(\)](#), then [interp\(\)](#) will return incorrect results.

No caching is done (except for the caching already present in `sm_interp`), and successive calls to [interp\(\)](#) with the same values of x and y will take nearly the same amount of time.

#### Todo

Test non-square data

#### Todo

Allow specification of data as `[ny][nx]` or as `[nx][ny]`?

---



**Todo**

Need to rewrite to use `o2scl_interp` after that class is finished.

**Todo**

Could also include mixed second/first derivatives: `deriv_xxy` and `deriv_xyy`.

Definition at line 60 of file `twod_intp.h`.

**Public Member Functions**

- `int set_data (int n_x, int n_y, ovector &x_fun, ovector &y_fun, omatrix &u_data, bool x_first=true)`  
*Initialize the data for the 2-dimensional interpolation.*
- `int reset_interp ()`  
*Reset the stored interpolation since the data has changed.*
- `double interp (double x, double y)`  
*Perform the 2-d interpolation.*
- `double deriv_x (double x, double y)`  
*Compute the partial derivative in the x-direction.*
- `double deriv2_x (double x, double y)`  
*Compute the partial second derivative in the x-direction.*
- `double integ_x (double x0, double x1, double y)`  
*Compute the integral in the x-direction between x=x0 and x=x1.*
- `double deriv_y (double x, double y)`  
*Compute the partial derivative in the y-direction.*
- `double deriv2_y (double x, double y)`  
*Compute the partial second derivative in the y-direction.*
- `double integ_y (double x, double y0, double y1)`  
*Compute the integral in the y-direction between y=y0 and y=y1.*
- `double deriv_xy (double x, double y)`  
*Compute the mixed partial derivative  $\frac{\partial^2 f}{\partial x \partial y}$ .*

**3.313.2 Member Function Documentation****3.313.2.1 `int set_data (int n_x, int n_y, ovector &x_fun, ovector &y_fun, omatrix &u_data, bool x_first = true)`**

Initialize the data for the 2-dimensional interpolation.

The interpolation type (passed directly to `int_type`) is specified in `int_type` and the data is specified in `data`. The data should be arranged so that the first array index is the y-value (the "row") and the second array index is the x-value (the "column"). The arrays `xfun` and `yfun` specify the two independent variables. `xfun` should be an array of length `nx`, and should be an array of length `ny`. The array `data` should be a two-dimensional array of size `[ny][nx]`.

If `x_first` is true, then `set_data()` creates interpolation objects for each of the rows. Calls to `interp()` then uses these to create a column at the specified value of `x`. An interpolation object is created at this column to find the value of the function at the specified value `y`. If `x_first` is false, the opposite strategy is employed. These two options will give different results (unless linear interpolation is explicitly selected). Using `x_first=false` will be slightly slower, since the data has to be rearranged in the opposite order first. In general, if the data is "more accurate" in the `x` direction than in the `y` direction, it is probably better to choose `x_first=true`.

Valid values of `int_type` are:

- 0 - default (cubic spline)
- 1 - linear
- 2 - cubic spline

- 3 - periodic cubic spline
- 4 - akima spline
- 5 - periodic akima spline

### 3.313.2.2 `int reset_interp ()`

Reset the stored interpolation since the data has changed.

This will return an error if the [set\\_data\(\)](#) has not been called

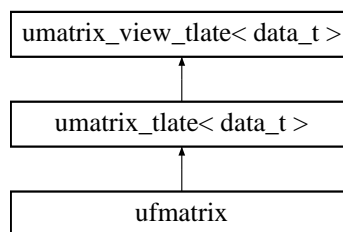
The documentation for this class was generated from the following file:

- [twod\\_intp.h](#)

## 3.314 **ufmatrix Class Template Reference**

```
#include <umatrix_tlate.h>
```

Inheritance diagram for `ufmatrix`::



### 3.314.1 Detailed Description

```
template<size_t N, size_t M> class ufmatrix< N, M >
```

A matrix where the memory allocation is performed in the constructor.

Definition at line 685 of file `umatrix_tlate.h`.

#### Public Member Functions

- [ufmatrix \(\)](#)

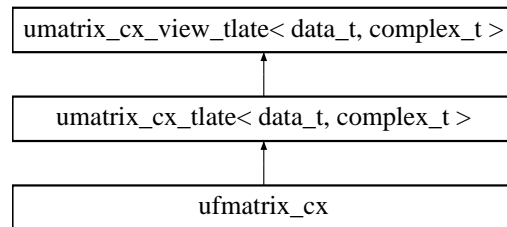
The documentation for this class was generated from the following file:

- [umatrix\\_tlate.h](#)

## 3.315 **ufmatrix\_cx Class Template Reference**

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for `ufmatrix_cx`::



### 3.315.1 Detailed Description

**template<size\_t N, size\_t M> class uvector\_cx< N, M >**

A matrix where the memory allocation is performed in the constructor.

Definition at line 696 of file uvector\_cx\_tlate.h.

#### Public Member Functions

- [uvector\\_cx \(\)](#)

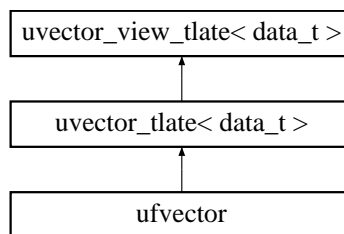
The documentation for this class was generated from the following file:

- [uvector\\_cx\\_tlate.h](#)

## 3.316 **ufvector Class Template Reference**

```
#include <ufvector_tlate.h>
```

Inheritance diagram for ufvector::



### 3.316.1 Detailed Description

**template<size\_t N = 0> class ufvector< N >**

A vector with unit-stride where the memory allocation is performed in the constructor.

Definition at line 795 of file ufvector\_tlate.h.

#### Public Member Functions

- [ufvector \(\)](#)

The documentation for this class was generated from the following file:

- [uvector\\_tlate.h](#)

## 3.317 `umatrix_alloc` Class Reference

```
#include <umatrix_tlate.h>
```

### 3.317.1 Detailed Description

A simple class to provide an `allocate()` function for `umatrix`.

Definition at line 673 of file `umatrix_tlate.h`.

#### Public Member Functions

- void `allocate` (`umatrix` &o, int i, int j)  
*Allocate  $\forall$  for  $i$  elements.*
- void `free` (`umatrix` &o)  
*Free memory.*

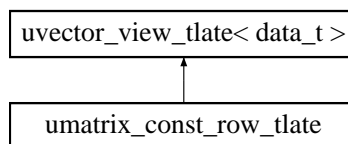
The documentation for this class was generated from the following file:

- [umatrix\\_tlate.h](#)

## 3.318 `umatrix_const_row_tlate` Class Template Reference

```
#include <umatrix_tlate.h>
```

Inheritance diagram for `umatrix_const_row_tlate`:



### 3.318.1 Detailed Description

```
template<class data_t> class umatrix_const_row_tlate< data_t >
```

Create a const vector from a row of a matrix.

Definition at line 593 of file `umatrix_tlate.h`.

#### Public Member Functions

- `umatrix_const_row_tlate` (const `umatrix_view_tlate`< data\_t > &m, size\_t i)  
*Create a vector from row  $i$  of matrix  $m$ .*

The documentation for this class was generated from the following file:

- [umatrix\\_tlate.h](#)

## 3.319 `umatrix_cx_alloc` Class Reference

```
#include <umatrix_cx_tlate.h>
```

### 3.319.1 Detailed Description

A simple class to provide an `allocate()` function for `umatrix_cx`.

Definition at line 684 of file `umatrix_cx_tlate.h`.

#### Public Member Functions

- void `allocate` (`umatrix_cx` &o, int i, int j)  
*Allocate  $\forall$  for i elements.*
- void `free` (`umatrix_cx` &o)  
*Free memory.*

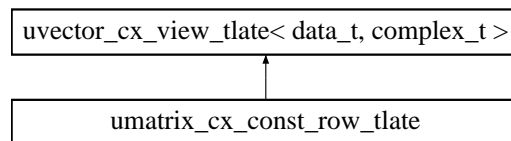
The documentation for this class was generated from the following file:

- [umatrix\\_cx\\_tlate.h](#)

## 3.320 `umatrix_cx_const_row_tlate` Class Template Reference

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for `umatrix_cx_const_row_tlate`:



### 3.320.1 Detailed Description

```
template<class data_t, class complex_t> class umatrix_cx_const_row_tlate< data_t, complex_t >
```

Create a const vector from a row of a matrix.

Definition at line 613 of file `umatrix_cx_tlate.h`.

#### Public Member Functions

- `umatrix_cx_const_row_tlate` (const `umatrix_cx_view_tlate`< data\_t, complex\_t > &m, size\_t i)  
*Create a vector from row i of matrix m.*

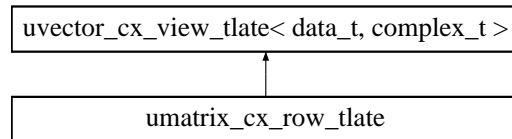
The documentation for this class was generated from the following file:

- [umatrix\\_cx\\_tlate.h](#)

### 3.321 `umatrix_cx_row_tlate` Class Template Reference

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for `umatrix_cx_row_tlate`::



#### 3.321.1 Detailed Description

```
template<class data_t, class complex_t> class umatrix_cx_row_tlate< data_t, complex_t >
```

Create a vector from a row of a matrix.

Definition at line 597 of file `umatrix_cx_tlate.h`.

#### Public Member Functions

- [umatrix\\_cx\\_row\\_tlate](#) ([umatrix\\_cx\\_view\\_tlate](#)< `data_t`, `complex_t` > &`m`, `size_t i`)  
Create a vector from row `i` of matrix `m`.

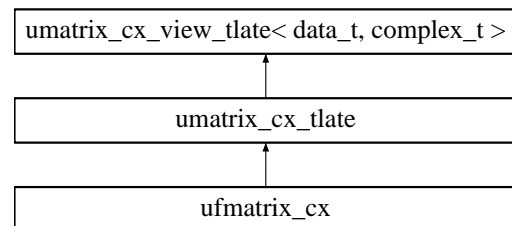
The documentation for this class was generated from the following file:

- [umatrix\\_cx\\_tlate.h](#)

### 3.322 `umatrix_cx_tlate` Class Template Reference

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for `umatrix_cx_tlate`::



#### 3.322.1 Detailed Description

```
template<class data_t, class complex_t> class umatrix_cx_tlate< data_t, complex_t >
```

A matrix of double-precision numbers.

Definition at line 363 of file `umatrix_cx_tlate.h`.

## Public Member Functions

- [~umatrix\\_cx\\_tlate](#) ()

### Standard constructor

- [umatrix\\_cx\\_tlate](#) (size\_t r=0, size\_t c=0)  
*Create an umatrix of size n with owner as 'true'.*

### Copy constructors

- [umatrix\\_cx\\_tlate](#) (const [umatrix\\_cx\\_tlate](#) &v)  
*Deep copy constructor; allocate new space and make a copy.*
- [umatrix\\_cx\\_tlate](#) (const [umatrix\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > &v)  
*Deep copy constructor; allocate new space and make a copy.*
- [umatrix\\_cx\\_tlate](#) & [operator=](#) (const [umatrix\\_cx\\_tlate](#) &v)  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*
- [umatrix\\_cx\\_tlate](#) & [operator=](#) (const [umatrix\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > &v)  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*
- [umatrix\\_cx\\_tlate](#) (size\_t n, [uvector\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > uva[ ])  
*Deep copy from an array of uvectors.*
- [umatrix\\_cx\\_tlate](#) (size\_t n, size\_t n2, data\_t \*\*csa)  
*Deep copy from a C-style 2-d array.*

### Memory allocation

- int [allocate](#) (size\_t nrows, size\_t ncols)  
*Allocate memory after freeing any memory presently in use.*
- int [free](#) ()  
*Free the memory.*

### Other methods

- [umatrix\\_cx\\_tlate](#)< data\_t, complex\_t > [transpose](#) ()

## 3.322.2 Member Function Documentation

### 3.322.2.1 int free () [inline]

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 568 of file [umatrix\\_cx\\_tlate.h](#).

### 3.322.2.2 umatrix\_cx\_tlate<data\_t,complex\_t> transpose () [inline]

Compute the transpose (even if matrix is not square)

Definition at line 583 of file [umatrix\\_cx\\_tlate.h](#).

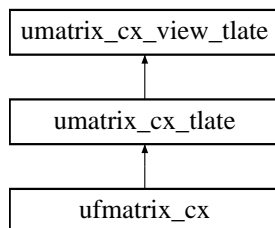
The documentation for this class was generated from the following file:

- [umatrix\\_cx\\_tlate.h](#)

### 3.323 `umatrix_cx_view_tlate` Class Template Reference

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for `umatrix_cx_view_tlate`:



#### 3.323.1 Detailed Description

```
template<class data_t, class complex_t> class umatrix_cx_view_tlate< data_t, complex_t >
```

A matrix view of complex numbers.

Definition at line 50 of file `umatrix_cx_tlate.h`.

#### Public Member Functions

- [~umatrix\\_cx\\_view\\_tlate](#) ()

#### Copy constructors

- [umatrix\\_cx\\_view\\_tlate](#) (const [umatrix\\_cx\\_view\\_tlate](#) &v)  
*So2scllow copy constructor - create a new view of the same matrix.*
- [umatrix\\_cx\\_view\\_tlate](#) & [operator=](#) (const [umatrix\\_cx\\_view\\_tlate](#) &v)  
*So2scllow copy constructor - create a new view of the same matrix.*

#### Get and set methods

- `complex_t * operator[ ]` (size\_t i)  
*Array-like indexing.*
- `const complex_t * operator[ ]` (size\_t i) const  
*Array-like indexing.*
- `complex_t & operator()` (size\_t i, size\_t j)  
*Array-like indexing.*
- `const complex_t & operator()` (size\_t i, size\_t j) const  
*Array-like indexing.*
- `complex_t get` (size\_t i, size\_t j) const  
*Get (with optional range-checking).*
- `complex_t * get\_ptr` (size\_t i, size\_t j)  
*Get pointer (with optional range-checking).*
- `const complex_t * get\_const\_ptr` (size\_t i, size\_t j) const  
*Get pointer (with optional range-checking).*
- `int set` (size\_t i, size\_t j, complex\_t val)  
*Set (with optional range-checking).*
- `int set` (size\_t i, size\_t j, data\_t re, data\_t im)  
*Set (with optional range-checking).*
- `int set\_all` (complex\_t val)  
*Set all of the value to be the value val.*
- `size_t rows` () const



*Method to return number of rows.*

- `size_t cols () const`

*Method to return number of columns.*

### Other methods

- `bool is_owner () const`

*Return true if this object owns the data it refers to.*

### Arithmetic

- `umatrix_cx_view_tlate< data_t, complex_t > & operator+= (const umatrix_cx_view_tlate< data_t, complex_t > &x)`  
*operator+=*
- `umatrix_cx_view_tlate< data_t, complex_t > & operator-= (const umatrix_cx_view_tlate< data_t, complex_t > &x)`  
*operator-=*
- `umatrix_cx_view_tlate< data_t, complex_t > & operator+= (const data_t &y)`  
*operator+=*
- `umatrix_cx_view_tlate< data_t, complex_t > & operator-= (const data_t &y)`  
*operator-=*
- `umatrix_cx_view_tlate< data_t, complex_t > & operator *= (const data_t &y)`  
*operator\*=*

### Protected Member Functions

- `umatrix_cx_view_tlate ()`

*Empty constructor provided for use by `umatrix_cx_tlate(const umatrix_cx_tlate &v)`.*

### Protected Attributes

- `data_t * data`  
*The data.*
- `size_t size1`  
*The number of rows.*
- `size_t size2`  
*The number of columns.*
- `int owner`  
*Zero if memory is owned elsewhere, 1 otherwise.*

## 3.323.2 Member Function Documentation

### 3.323.2.1 `size_t rows () const` [inline]

Method to return number of rows.

If no memory has been allocated, this will quietly return zero.

Definition at line 254 of file `umatrix_cx_tlate.h`.

### 3.323.2.2 `size_t cols () const` [inline]

Method to return number of columns.

If no memory has been allocated, this will quietly return zero.

Definition at line 264 of file `umatrix_cx_tlate.h`.

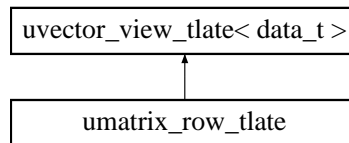
The documentation for this class was generated from the following file:

- `umatrix_cx_tlate.h`

## 3.324 umatrix\_row\_tlate Class Template Reference

```
#include <umatrix_tlate.h>
```

Inheritance diagram for umatrix\_row\_tlate::



### 3.324.1 Detailed Description

```
template<class data_t> class umatrix_row_tlate< data_t >
```

Create a vector from a row of a matrix.

Definition at line 577 of file umatrix\_tlate.h.

#### Public Member Functions

- [umatrix\\_row\\_tlate](#) ([umatrix\\_view\\_tlate](#)< data\_t > &m, size\_t i)  
*Create a vector from row i of matrix m.*

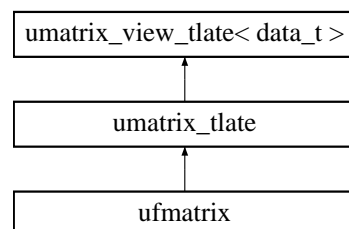
The documentation for this class was generated from the following file:

- [umatrix\\_tlate.h](#)

## 3.325 umatrix\_tlate Class Template Reference

```
#include <umatrix_tlate.h>
```

Inheritance diagram for umatrix\_tlate::



### 3.325.1 Detailed Description

```
template<class data_t> class umatrix_tlate< data_t >
```

A matrix of double-precision numbers.

Definition at line 344 of file umatrix\_tlate.h.

## Public Member Functions

- [~umatrix\\_tlate](#) ()

### Standard constructor

- [umatrix\\_tlate](#) (size\_t r=0, size\_t c=0)  
*Create an umatrix of size n with owner as 'true'.*

### Copy constructors

- [umatrix\\_tlate](#) (const [umatrix\\_tlate](#) &v)  
*Deep copy constructor; allocate new space and make a copy.*
- [umatrix\\_tlate](#) (const [umatrix\\_view\\_tlate](#)< data\_t > &v)  
*Deep copy constructor; allocate new space and make a copy.*
- [umatrix\\_tlate](#) & [operator=](#) (const [umatrix\\_tlate](#) &v)  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*
- [umatrix\\_tlate](#) & [operator=](#) (const [umatrix\\_view\\_tlate](#)< data\_t > &v)  
*Deep copy constructor; if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*
- [umatrix\\_tlate](#) (size\_t n, [uvector\\_view\\_tlate](#)< data\_t > uva[ ])  
*Deep copy from an array of uvectors.*
- [umatrix\\_tlate](#) (size\_t n, size\_t n2, data\_t \*\*csa)  
*Deep copy from a C-style 2-d array.*

### Memory allocation

- int [allocate](#) (size\_t nrows, size\_t ncols)  
*Allocate memory after freeing any memory presently in use.*
- int [free](#) ()  
*Free the memory.*

### Other methods

- [umatrix\\_tlate](#)< data\_t > [transpose](#) ()

## 3.325.2 Member Function Documentation

### 3.325.2.1 int free () [inline]

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 548 of file [umatrix\\_tlate.h](#).

### 3.325.2.2 umatrix\_tlate<data\_t> transpose () [inline]

Compute the transpose (even if matrix is not square)

Definition at line 563 of file [umatrix\\_tlate.h](#).

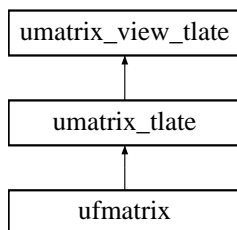
The documentation for this class was generated from the following file:

- [umatrix\\_tlate.h](#)

## 3.326 `umatrix_view_tlate` Class Template Reference

```
#include <umatrix_tlate.h>
```

Inheritance diagram for `umatrix_view_tlate`:



### 3.326.1 Detailed Description

```
template<class data_t> class umatrix_view_tlate< data_t >
```

A matrix view of double-precision numbers.

Definition at line 49 of file `umatrix_tlate.h`.

#### Public Member Functions

- [~umatrix\\_view\\_tlate](#) ()

#### Copy constructors

- [umatrix\\_view\\_tlate](#) (const [umatrix\\_view\\_tlate](#) &v)  
*So2scflow copy constructor - create a new view of the same matrix.*
- [umatrix\\_view\\_tlate](#) & [operator=](#) (const [umatrix\\_view\\_tlate](#) &v)  
*So2scflow copy constructor - create a new view of the same matrix.*

#### Get and set methods

- `data_t * operator\[ \] (size_t i)`  
*Array-like indexing.*
- `const data_t * operator\[ \] (size_t i) const`  
*Array-like indexing.*
- `data_t & operator\(\) (size_t i, size_t j)`  
*Array-like indexing.*
- `const data_t & operator\(\) (size_t i, size_t j) const`  
*Array-like indexing.*
- `data_t get (size_t i, size_t j) const`  
*Get (with optional range-checking).*
- `data_t * get\_ptr (size_t i, size_t j)`  
*Get pointer (with optional range-checking).*
- `const data_t * get\_const\_ptr (size_t i, size_t j) const`  
*Get pointer (with optional range-checking).*
- `int set (size_t i, size_t j, data_t val)`  
*Set (with optional range-checking).*
- `int set\_all (double val)`  
*Set all of the value to be the value val.*
- `size_t rows () const`  
*Method to return number of rows.*
- `size_t cols () const`  
*Method to return number of columns.*

## Other methods

- `bool is_owner () const`  
Return true if this object owns the data it refers to.

## Arithmetic

- `umatrix_view_tlate< data_t > & operator+= (const umatrix_view_tlate< data_t > &x)`  
`operator+=`
- `umatrix_view_tlate< data_t > & operator-= (const umatrix_view_tlate< data_t > &x)`  
`operator-=`
- `umatrix_view_tlate< data_t > & operator+= (const data_t &y)`  
`operator+=`
- `umatrix_view_tlate< data_t > & operator-= (const data_t &y)`  
`operator-=`
- `umatrix_view_tlate< data_t > & operator *= (const data_t &y)`  
`operator*=`

## Protected Member Functions

- `umatrix_view_tlate ()`  
Empty constructor provided for use by `umatrix_tlate(const umatrix_tlate &v)`.

## Protected Attributes

- `data_t * data`  
The data.
- `size_t size1`  
The number of rows.
- `size_t size2`  
The number of columns.
- `int owner`  
Zero if memory is owned elsewhere, 1 otherwise.

### 3.326.2 Member Function Documentation

#### 3.326.2.1 `size_t rows () const` [inline]

Method to return number of rows.

If no memory has been allocated, this will quietly return zero.

Definition at line 235 of file `umatrix_tlate.h`.

#### 3.326.2.2 `size_t cols () const` [inline]

Method to return number of columns.

If no memory has been allocated, this will quietly return zero.

Definition at line 245 of file `umatrix_tlate.h`.

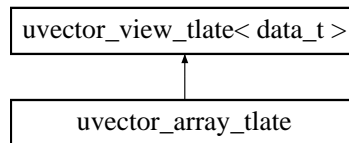
The documentation for this class was generated from the following file:

- `umatrix_tlate.h`

### 3.327 `uvector_array_tlate` Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_array_tlate`:



#### 3.327.1 Detailed Description

```
template<class data_t> class uvector_array_tlate< data_t >
```

Create a vector from an array.

Definition at line 594 of file `uvector_tlate.h`.

#### Public Member Functions

- [`uvector\_array\_tlate`](#) (`size_t n`, `data_t *dat`)  
Create a vector from `dat` with size `n`.

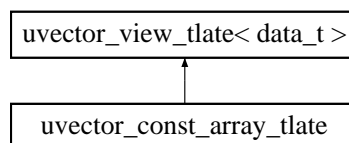
The documentation for this class was generated from the following file:

- [uvector\\_tlate.h](#)

### 3.328 `uvector_const_array_tlate` Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_const_array_tlate`:



#### 3.328.1 Detailed Description

```
template<class data_t> class uvector_const_array_tlate< data_t >
```

Create a vector from an const array.

Definition at line 628 of file `uvector_tlate.h`.

#### Public Member Functions

- [`uvector\_const\_array\_tlate`](#) (`size_t n`, `const data_t *dat`)

Create a vector from `dat` with size `n`.

- [~uvector\\_const\\_array\\_tlate\(\)](#)

## Protected Member Functions

### Ensure `\c` const by hiding non-const members

- `data_t & operator[] (size_t i)`  
*Array-like indexing.*
- `data_t & operator() (size_t i)`  
*Array-like indexing.*
- `data_t * get_ptr (size_t i)`  
*Get pointer (with optional range-checking).*
- `int set (size_t i, data_t val)`  
*Set (with optional range-checking).*
- `int swap (uvector_view_tlate< data_t > &x)`  
*Swap vectors.*
- `int set_all (double val)`
- `uvector_view_tlate< data_t > & operator+= (const uvector_view_tlate< data_t > &x)`  
*operator+=*
- `uvector_view_tlate< data_t > & operator-= (const uvector_view_tlate< data_t > &x)`  
*operator-=*
- `uvector_view_tlate< data_t > & operator *= (const data_t &y)`  
*operator\*=*

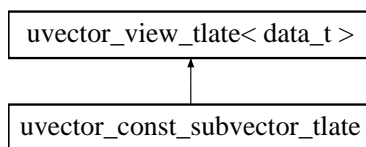
The documentation for this class was generated from the following file:

- [uvector\\_tlate.h](#)

## 3.329 `uvector_const_subvector_tlate` Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_const_subvector_tlate`:



### 3.329.1 Detailed Description

```
template<class data_t> class uvector_const_subvector_tlate< data_t >
```

Create a const vector from a subvector of another vector.

Definition at line 678 of file `uvector_tlate.h`.

## Public Member Functions

- [uvector\\_const\\_subvector\\_tlate](#) (const [uvector\\_view\\_tlate](#)< data\_t > &orig, size\_t offset, size\_t n)  
*Create a vector from orig.*

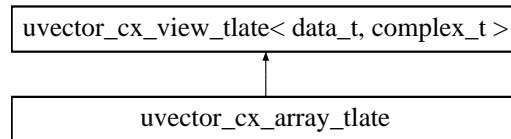
The documentation for this class was generated from the following file:

- [uvector\\_tlate.h](#)

### 3.330 `uvector_cx_array_tlate` Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_array_tlate`:



#### 3.330.1 Detailed Description

```
template<class data_t, class complex_t> class uvector_cx_array_tlate< data_t, complex_t >
```

Create a vector from an array.

Definition at line 505 of file `uvector_cx_tlate.h`.

#### Public Member Functions

- [`uvector\_cx\_array\_tlate`](#) (`size_t n`, `data_t *dat`)  
Create a vector from `dat` with size `n`.

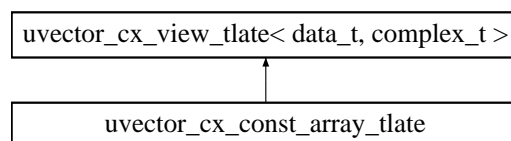
The documentation for this class was generated from the following file:

- [`uvector\_cx\_tlate.h`](#)

### 3.331 `uvector_cx_const_array_tlate` Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_const_array_tlate`:



#### 3.331.1 Detailed Description

```
template<class data_t, class complex_t> class uvector_cx_const_array_tlate< data_t, complex_t >
```

Create a vector from an array.

Definition at line 539 of file `uvector_cx_tlate.h`.

#### Public Member Functions

- [`uvector\_cx\_const\_array\_tlate`](#) (`size_t n`, `const data_t *dat`)



Create a vector from `dat` with size `n`.

- [~uvector\\_cx\\_const\\_array\\_tlate](#) ()

### Protected Member Functions

These are inaccessible to ensure the vector is `\c const`.

- `data_t & operator[]` (size\_t i)  
*Array-like indexing.*
- `data_t & operator()` (size\_t i)  
*Array-like indexing.*
- `data_t * get_ptr` (size\_t i)  
*Get pointer (with optional range-checking).*
- `int set` (size\_t i, data\_t val)
- `int swap` ([uvector\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > &x)  
*Swap vectors.*
- `int set_all` (double val)
- `uvector_cx_view_tlate`< data\_t, complex\_t > & `operator+=` (const [uvector\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > &x)  
*operator+=*
- `uvector_cx_view_tlate`< data\_t, complex\_t > & `operator-=` (const [uvector\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > &x)  
*operator-=*
- `uvector_cx_view_tlate`< data\_t, complex\_t > & `operator *=` (const data\_t &y)  
*operator\*=*

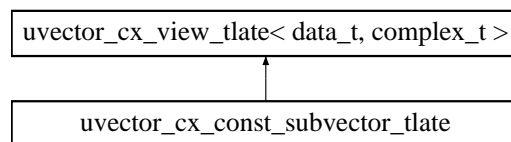
The documentation for this class was generated from the following file:

- [uvector\\_cx\\_tlate.h](#)

## 3.332 `uvector_cx_const_subvector_tlate` Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_const_subvector_tlate`:



### 3.332.1 Detailed Description

```
template<class data_t, class complex_t> class uvector_cx_const_subvector_tlate< data_t, complex_t >
```

Create a vector from a subvector of another.

Definition at line 590 of file `uvector_cx_tlate.h`.

### Public Member Functions

- `uvector_cx_const_subvector_tlate` (const [uvector\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > &orig, size\_t offset, size\_t n)  
*Create a vector from orig.*

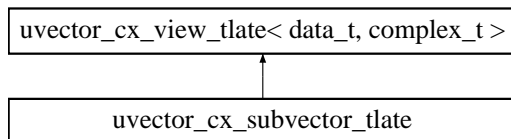
The documentation for this class was generated from the following file:

- [uvector\\_cx\\_tlate.h](#)

### 3.333 `uvector_cx_subvector_tlate` Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_subvector_tlate`:



#### 3.333.1 Detailed Description

```
template<class data_t, class complex_t> class uvector_cx_subvector_tlate< data_t, complex_t >
```

Create a vector from a subvector of another.

Definition at line 520 of file `uvector_cx_tlate.h`.

#### Public Member Functions

- [`uvector\_cx\_subvector\_tlate`](#) ([`uvector\_cx\_view\_tlate`](#)< `data_t`, `complex_t` > &`orig`, `size_t` `offset`, `size_t` `n`)  
Create a vector from `orig`.

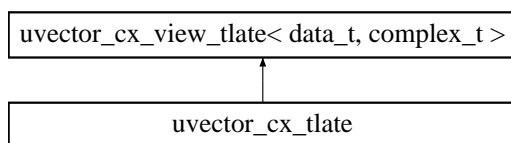
The documentation for this class was generated from the following file:

- [uvector\\_cx\\_tlate.h](#)

### 3.334 `uvector_cx_tlate` Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_tlate`:



#### 3.334.1 Detailed Description

```
template<class data_t, class complex_t> class uvector_cx_tlate< data_t, complex_t >
```

A vector of double-precision numbers with unit stride.

There are several global binary operators associated with objects of type [`uvector\_cx\_tlate`](#). The are documented in the "Functions" section of [uvector\\_cx\\_tlate.h](#).

Definition at line 336 of file `uvector_cx_tlate.h`.

## Public Member Functions

- [~uvector\\_cx\\_tlate](#) ()

## Standard constructor

- [uvector\\_cx\\_tlate](#) (size\_t n=0)  
*Create an `uvector_cx` of size `n` with owner as 'true'.*

## Copy constructors

- [uvector\\_cx\\_tlate](#) (const [uvector\\_cx\\_tlate](#) &v)  
*Deep copy constructor - allocate new space and make a copy.*
- [uvector\\_cx\\_tlate](#) (const [uvector\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > &v)  
*Deep copy constructor - allocate new space and make a copy.*
- [uvector\\_cx\\_tlate](#) & operator= (const [uvector\\_cx\\_tlate](#) &v)  
*Deep copy constructor - if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*
- [uvector\\_cx\\_tlate](#) & operator= (const [uvector\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > &v)  
*Deep copy constructor - if owner is true, allocate space and make a new copy, otherwise, just copy into the view.*

## Memory allocation

- int [allocate](#) (size\_t nsize)  
*Allocate memory for size `n` after freeing any memory presently in use.*
- int [free](#) ()  
*Free the memory.*

### 3.334.2 Member Function Documentation

#### 3.334.2.1 int free () [inline]

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 490 of file `uvector_cx_tlate.h`.

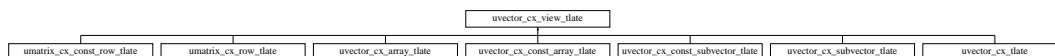
The documentation for this class was generated from the following file:

- [uvector\\_cx\\_tlate.h](#)

## 3.335 `uvector_cx_view_tlate` Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_view_tlate`:



### 3.335.1 Detailed Description

```
template<class data_t, class complex_t> class uvector_cx_view_tlate< data_t, complex_t >
```

A vector view of complex numbers with unit stride.

**Object status:** Experimental

## Todo

Write `lookup()` method, and possible an `erase()` method.

Definition at line 50 of file `uvector_cx_tlate.h`.

## Public Member Functions

- `~uvector_cx_view_tlate()`

### Copy constructors

- `uvector_cx_view_tlate` (const `uvector_cx_view_tlate` &`v`)  
*Copy constructor - create a new view of the same vector.*
- `uvector_cx_view_tlate` & `operator=` (const `uvector_cx_view_tlate` &`v`)  
*Copy constructor - create a new view of the same vector.*

### Get and set methods

- `complex_t` & `operator[]` (`size_t` `i`)  
*Array-like indexing.*
- const `complex_t` & `operator[]` (`size_t` `i`) const  
*Array-like indexing.*
- `complex_t` & `operator()` (`size_t` `i`)  
*Array-like indexing.*
- const `complex_t` & `operator()` (`size_t` `i`) const  
*Array-like indexing.*
- `complex_t` `get` (`size_t` `i`) const  
*Get (with optional range-checking).*
- `complex_t` \* `get_ptr` (`size_t` `i`)  
*Get pointer (with optional range-checking).*
- const `complex_t` \* `get_const_ptr` (`size_t` `i`) const  
*Get pointer (with optional range-checking).*
- int `set` (`size_t` `i`, const `complex_t` &`val`)  
*Set (with optional range-checking).*
- int `set_all` (const `complex_t` &`val`)  
*Set all of the value to be the value `val`.*
- `size_t` `size` () const  
*Method to return vector size.*

### Other methods

- int `swap` (`uvector_cx_view_tlate`< `data_t`, `complex_t` > &`x`)  
*Swap vectors.*
- bool `is_owner` () const  
*Return true if this object owns the data it refers to.*

### Arithmetic

- `uvector_cx_view_tlate`< `data_t`, `complex_t` > & `operator+=` (const `uvector_cx_view_tlate`< `data_t`, `complex_t` > &`x`)  
*operator+=*
- `uvector_cx_view_tlate`< `data_t`, `complex_t` > & `operator-=` (const `uvector_cx_view_tlate`< `data_t`, `complex_t` > &`x`)  
*operator-=*
- `uvector_cx_view_tlate`< `data_t`, `complex_t` > & `operator+=` (const `data_t` &`y`)  
*operator+=*
- `uvector_cx_view_tlate`< `data_t`, `complex_t` > & `operator-=` (const `data_t` &`y`)  
*operator-=*
- `uvector_cx_view_tlate`< `data_t`, `complex_t` > & `operator*=` (const `data_t` &`y`)  
*operator\*=*
- `data_t` `norm` () const  
*Norm.*

## Protected Member Functions

- [`uvector\_cx\_view\_tlate`](#) ()  
*Empty constructor provided for use by `uvector_cx_tlate(const uvector_cx_tlate &v)`.*

## Protected Attributes

- `data_t * data`  
*The data.*
- `size_t sz`  
*The vector sz.*
- `int owner`  
*Zero if memory is owned elsewhere, 1 otherwise.*

## 3.335.2 Member Function Documentation

### 3.335.2.1 `size_t size () const` [inline]

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 228 of file `uvector_cx_tlate.h`.

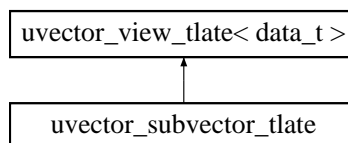
The documentation for this class was generated from the following file:

- [`uvector\_cx\_tlate.h`](#)

## 3.336 `uvector_subvector_tlate` Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_subvector_tlate`:



### 3.336.1 Detailed Description

```
template<class data_t> class uvector_subvector_tlate< data_t >
```

Create a vector from a subvector of another.

Definition at line 609 of file `uvector_tlate.h`.

## Public Member Functions

- [`uvector\_subvector\_tlate`](#) ([`uvector\_view\_tlate`](#)< `data_t` > &orig, `size_t` offset, `size_t` n)  
*Create a vector from orig.*

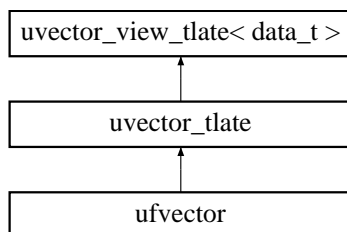
The documentation for this class was generated from the following file:

- [`uvector\_tlate.h`](#)

### 3.337 `uvector_tlate` Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_tlate`::



#### 3.337.1 Detailed Description

**template<class data\_t> class `uvector_tlate`< data\_t >**

A vector with unit stride.

There are several global binary operators associated with objects of type `uvector_tlate`. They are documented in the "Functions" section of [uvector\\_tlate.h](#).

Definition at line 386 of file `uvector_tlate.h`.

#### Public Member Functions

- [~uvector\\_tlate](#) ()

#### Standard constructor

- [uvector\\_tlate](#) (size\_t n=0)  
Create an uvector of size n with owner as 'true'.

#### Copy constructors

- [uvector\\_tlate](#) (const [uvector\\_tlate](#) &v)  
Deep copy constructor - allocate new space and make a copy.
- [uvector\\_tlate](#) (const [uvector\\_view\\_tlate](#)< data\_t > &v)  
Deep copy constructor - allocate new space and make a copy.
- [uvector\\_tlate](#) & [operator=](#) (const [uvector\\_tlate](#) &v)  
Deep copy constructor - if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- [uvector\\_tlate](#) & [operator=](#) (const [uvector\\_view\\_tlate](#)< data\_t > &v)  
Deep copy constructor - if owner is true, allocate space and make a new copy, otherwise, just copy into the view.

#### Memory allocation

- int [allocate](#) (size\_t nsize)  
Allocate memory for size n after freeing any memory presently in use.
- int [free](#) ()  
Free the memory.

#### Other methods

- int [erase](#) (size\_t ix)  
Erase an element from the array.

### 3.337.2 Member Function Documentation

#### 3.337.2.1 `int free ()` `[inline]`

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 546 of file `uvector_tlate.h`.

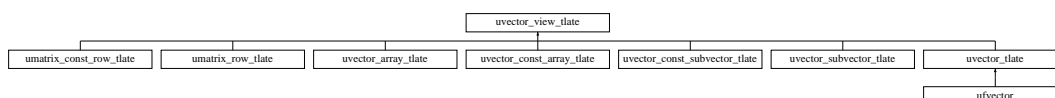
The documentation for this class was generated from the following file:

- [uvector\\_tlate.h](#)

## 3.338 `uvector_view_tlate` Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_view_tlate::`



### 3.338.1 Detailed Description

```
template<class data_t> class uvector_view_tlate< data_t >
```

A vector view with unit stride.

Definition at line 47 of file `uvector_tlate.h`.

#### Public Member Functions

- [~uvector\\_view\\_tlate \(\)](#)

#### Copy constructors

- [uvector\\_view\\_tlate \(const uvector\\_view\\_tlate &v\)](#)  
*Copy constructor - create a new view of the same vector.*
- [uvector\\_view\\_tlate & operator= \(const uvector\\_view\\_tlate &v\)](#)  
*Copy constructor - create a new view of the same vector.*

#### Get and set methods

- `data_t & operator[] (size_t i)`  
*Array-like indexing.*
- `const data_t & operator[] (size_t i) const`  
*Array-like indexing.*
- `data_t & operator() (size_t i)`  
*Array-like indexing.*
- `const data_t & operator() (size_t i) const`  
*Array-like indexing.*
- `data_t get (size_t i) const`  
*Get (with optional range-checking).*
- `data_t * get_ptr (size_t i)`  
*Get pointer (with optional range-checking).*

- `const data_t * get_const_ptr (size_t i) const`  
*Get pointer (with optional range-checking).*
- `int set (size_t i, data_t val)`  
*Set (with optional range-checking).*
- `int set_all (data_t val)`  
*Set all of the value to be the value `val`.*
- `size_t size () const`  
*Method to return vector size.*

### Other methods

- `int swap (uvector_view_tlate< data_t > &x)`  
*Swap vectors.*
- `bool is_owner () const`  
*Return true if this object owns the data it refers to.*
- `size_t lookup (const data_t x0) const`  
*Exhaustively look through the array for a particular value.*
- `data_t max () const`  
*Find the maximum element.*
- `data_t min () const`  
*Find the minimum element.*

### Arithmetic

- `uvector_view_tlate< data_t > & operator+= (const uvector_view_tlate< data_t > &x)`  
*operator+=*
- `uvector_view_tlate< data_t > & operator-= (const uvector_view_tlate< data_t > &x)`  
*operator-=*
- `uvector_view_tlate< data_t > & operator+= (const data_t &y)`  
*operator+=*
- `uvector_view_tlate< data_t > & operator-= (const data_t &y)`  
*operator-=*
- `uvector_view_tlate< data_t > & operator *= (const data_t &y)`  
*operator\*=*
- `data_t norm () const`  
*Norm.*

### Protected Member Functions

- `uvector_view_tlate ()`  
*Empty constructor provided for use by `uvector_tlate(const uvector_tlate &v)`.*

### Protected Attributes

- `data_t * data`  
*The data.*
- `size_t sz`  
*The vector `sz`.*
- `int owner`  
*Zero if memory is owned elsewhere, 1 otherwise.*

## 3.338.2 Member Function Documentation

### 3.338.2.1 `size_t size () const` [inline]

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 218 of file `uvector_tlate.h`.



### 3.338.2.2 size\_t lookup (const data\_t x0) const [inline]

Exhaustively look through the array for a particular value.

This can only fail if *all* of the entries in the array are not finite, in which case it calls [set\\_err\(\)](#) and returns 0. The error handler is reset at the beginning of [lookup\(\)](#).

Definition at line 260 of file [uvector\\_tlate.h](#).

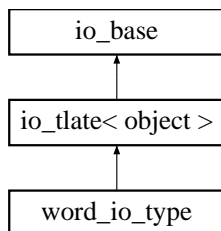
The documentation for this class was generated from the following file:

- [uvector\\_tlate.h](#)

## 3.339 word\_io\_type Class Reference

```
#include <collection.h>
```

Inheritance diagram for word\_io\_type::



### 3.339.1 Detailed Description

I/O object for words.

Definition at line 1834 of file [collection.h](#).

#### Public Member Functions

- [word\\_io\\_type](#) (const char \*t)  
*Desc.*
- [word\\_io\\_type](#) ()
- int [input](#) ([cinput](#) \*co, [in\\_file\\_format](#) \*ins, std::string \*dp)  
*Desc.*
- int [output](#) ([coutput](#) \*co, [out\\_file\\_format](#) \*outs, std::string \*dp)  
*Desc.*
- int [addw](#) ([collection](#) &co, std::string name, std::string w, bool overwrt=true)  
*Add a string to a [collection](#).*
- std::string [getw](#) ([collection](#) &co, std::string tname)  
*Get a word from a [collection](#).*
- int [get\\_def](#) ([collection](#) &co, std::string tname, std::string &op, std::string def="")  
*Get a word from a [collection](#).*
- const char \* [type](#) ()  
*Desc.*

The documentation for this class was generated from the following file:

- [collection.h](#)

## 4 O2scl File Documentation

### 4.1 array.h File Reference

#### 4.1.1 Detailed Description

Various array classes.

This file contains the three alloction classes

- [array\\_alloc](#)
- [array\\_2d\\_alloc](#)
- [pointer\\_alloc](#) the classes for the manipulation of array in [smart\\_interp](#)
- [array\\_reverse](#)
- [array\\_subvector](#)
- [array\\_subvector\\_reverse](#)
- [array\\_const\\_reverse](#)
- [array\\_const\\_subvector](#)
- [array\\_const\\_subvector\\_reverse](#) the array equivalent of `omatrix_row` or `umatrix_row` (see usage proposed in `src/ode/ode_it_solve_ts.cpp`)
- [array\\_row](#) and an array output function
- `array_out()`

#### Todo

Ensure that [array\\_row](#) works, either here or in `src/ode/ode_it_solve_ts.cpp`

Definition in file [array.h](#).

```
#include <iostream>
#include <cmath>
#include <string>
#include <fstream>
#include <sstream>
#include <o2scl/err_hnd.h>
#include <gsl/gsl_ieee_utils.h>
#include <gsl/gsl_sort.h>
```

#### Data Structures

- class [array\\_alloc](#)  
A simple class to provide an `allocate()` function for arrays.
- class [array\\_2d\\_alloc](#)  
A simple class to provide an `allocate()` function for 2-dimensional arrays.
- class [pointer\\_alloc](#)  
A simple class to provide an `allocate()` function for pointers.

- class [array\\_reverse](#)  
*A simple class which reverses the order of an array.*
- class [array\\_const\\_reverse](#)  
*A simple class which reverses the order of an array.*
- class [array\\_subvector](#)  
*A simple subvector class for an array (without error checking).*
- class [array\\_const\\_subvector](#)  
*A simple subvector class for a const array (without error checking).*
- class [array\\_subvector\\_reverse](#)  
*Reverse a subvector of an array.*
- class [array\\_const\\_subvector\\_reverse](#)  
*Reverse a subvector of a const array.*
- class [array\\_row](#)  
*Extract a row of a C-style 2d-array.*

## Functions

- `template<class vec_t>`  
`int vector\_out (std::ostream &os, size_t n, vec_t &v, bool endlne=false)`  
*Output a vector to a stream.*
- `template<class data_t, class vec_t>`  
`void sort\_downheap (vec_t &data, const size_t N, size_t k)`  
*Provide a downheap() function for [vector\\_sort](#)().*
- `template<class data_t, class vec_t>`  
`int vector\_sort (const size_t n, vec_t &data)`  
*Sort a vector.*
- `template<class data_t, class vec_t>`  
`int vector\_rotate (const size_t n, vec_t &data, size_t k)`  
*"Rotate" a vector so that the kth element is now the beginning*
- `template<class data_t, class vec_t>`  
`int vector\_max (const size_t n, vec_t &data, data_t &max, size_t &ix)`  
*Compute the maximum of the first n elements of a vector.*
- `template<class data_t, class vec_t>`  
`int vector\_min (const size_t n, vec_t &data, data_t &min, size_t &ix)`  
*Compute the minimum of the first n elements of a vector.*
- `template<class data_t, class vec_t>`  
`int vector\_sum (const size_t n, vec_t &data, data_t &sum)`  
*Compute the sum of the first n elements of a vector.*
- `template<class data_t, class vec_t>`  
`int vector\_avg (const size_t n, vec_t &data, data_t &avg)`  
*Compute the average of the first n elements of a vector.*
- `template<class data_t, class vec_t>`  
`int vector\_variance (const size_t n, vec_t &data, data_t &mean, data_t &var)`  
*Compute the variance of the first n elements of a vector given the mean mean.*
- `template<class data_t, class vec_t>`  
`int vector\_reverse (const size_t n, vec_t &data)`  
*Reverse a vector.*
- `template<class data_t, class vec_t>`  
`int vector\_stdev (const size_t n, vec_t &data, data_t &var)`  
*Compute the standard deviation of the first n elements of a vector.*

## 4.1.2 Function Documentation

### 4.1.2.1 `int vector_avg (const size_t n, vec_t & data, data_t & avg)` [inline]

Compute the average of the first n elements of a vector.

---

If `n` is zero, this will set `avg` to zero and return `gsl_success`.

Definition at line 511 of file `array.h`.

#### 4.1.2.2 `int vector_out (std::ostream &os, size_t n, vec_t &v, bool newline = false)` [inline]

Output a vector to a stream.

No trailing space is output after the last element, and an `newline` is output only if `newline` is set to `true`. If the parameter `n` is zero, this function silently does nothing.

Definition at line 380 of file `array.h`.

#### 4.1.2.3 `int vector_sum (const size_t n, vec_t &data, data_t &sum)` [inline]

Compute the sum of the first `n` elements of a vector.

If `n` is zero, this will set `avg` to zero and return `gsl_success`.

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

#### 4.1.2.4 `int vector_variance (const size_t n, vec_t &data, data_t &mean, data_t &var)` [inline]

Compute the variance of the first `n` elements of a vector given the mean `mean`.

If `n` is zero, this will set `avg` to zero and return `gsl_success`.

Definition at line 528 of file `array.h`.

## 4.2 columnify.h File Reference

### 4.2.1 Detailed Description

Functions to create output in columns.

Definition in file `columnify.h`.

```
#include <iostream>
#include <string>
#include <vector>
#include <o2scl/misc.h>
```

### Data Structures

- class `columnify`  
Create nicely formatted columns from a *table* of strings.

### Functions

- `template<class mat_t>`  
`int matrix_out (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)`  
A operator for naive matrix output.
- `template<class mat_t>`  
`int matrix_cx_out (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)`  
A operator for naive complex matrix output.
- `template<class mat_t>`  
`int array_2d_out (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)`

*A operator for naive 2-d array output.*

## 4.2.2 Function Documentation

### 4.2.2.1 int array\_2d\_out (std::ostream & os, mat\_t & A, size\_t nrows, size\_t ncols) [inline]

A operator for naive 2-d array output.

The type `mat_t` can be any 2d-array type which allows individual element access using `[size_t][size_t]`

This outputs all of the matrix elements using output settings specified by `os`. The alignment performed by `columnify` using `columnify::align_dp`, i.e. the numbers are aligned by their decimal points. If the numbers have no decimal points, then the decimal point is assumed to be to the right of the last character in the string representation of the number.

#### Todo

If all of the matrix elements are positive integers and scientific mode is not set, then we can avoid printing the extra spaces.

Definition at line 306 of file `columnify.h`.

### 4.2.2.2 int matrix\_out (std::ostream & os, mat\_t & A, size\_t nrows, size\_t ncols) [inline]

A operator for naive matrix output.

The type `mat_t` can be any matrix type which allows individual element access using `operator() (size_t, size_t)`.

This outputs all of the matrix elements using output settings specified by `os`. The alignment performed by `columnify` using `columnify::align_dp`, i.e. the numbers are aligned by their decimal points. If the numbers have no decimal points, then the decimal point is assumed to be to the right of the last character in the string representation of the number.

#### Todo

Might need to test to see what happens if all of the matrix elements are positive integers and scientific mode is not set.

Definition at line 233 of file `columnify.h`.

## 4.3 cx\_arith.h File Reference

### 4.3.1 Detailed Description

Complex arithmetic.

#### Todo

Define operators with assignment for complex + double

#### Todo

Ensure all the trig functions are tested

Definition in file `cx_arith.h`.

```
#include <iostream>
#include <complex>
#include <cmath>
#include <gsl/gsl_complex.h>
#include <gsl/gsl_complex_math.h>
```

## Namespaces

- namespace [o2scl\\_arith](#)

## Functions

### Binary operators for two complex numbers

- `gsl_complex operator+` (`gsl_complex x`, `gsl_complex y`)  
*Add two complex numbers.*
- `gsl_complex operator-` (`gsl_complex x`, `gsl_complex y`)  
*Subtract two complex numbers.*
- `gsl_complex operator *` (`gsl_complex x`, `gsl_complex y`)  
*Multiply two complex numbers.*
- `gsl_complex operator/` (`gsl_complex x`, `gsl_complex y`)  
*Divide two complex numbers.*

### Binary operators with assignment for two complex numbers

- `gsl_complex operator+=` (`gsl_complex &x`, `gsl_complex y`)  
*Add a complex number.*
- `gsl_complex operator-=` (`gsl_complex &x`, `gsl_complex y`)  
*Subtract a complex number.*
- `gsl_complex operator *=` (`gsl_complex &x`, `gsl_complex y`)  
*Multiply a complex number.*
- `gsl_complex operator/=` (`gsl_complex &x`, `gsl_complex y`)  
*Divide a complex number.*

### Binary operators with assignment for a complex and real

- `gsl_complex operator+` (`gsl_complex x`, `double y`)  
*Add a complex and real number.*
- `gsl_complex operator+` (`double y`, `gsl_complex x`)  
*Add a complex and real number.*
- `gsl_complex operator-` (`gsl_complex x`, `double y`)  
*Subtract a complex and real number.*
- `gsl_complex operator-` (`double y`, `gsl_complex x`)  
*Subtract a complex and real number.*
- `gsl_complex operator *` (`gsl_complex x`, `double y`)  
*Multiply a complex and real number.*
- `gsl_complex operator *` (`double y`, `gsl_complex x`)  
*Multiply a complex and real number.*
- `gsl_complex operator/` (`gsl_complex x`, `double y`)  
*Divide a complex and real number.*

### Miscellaneous functions

- `double arg` (`gsl_complex x`)
- `double abs` (`gsl_complex x`)
- `double abs2` (`gsl_complex z`)
- `gsl_complex conjugate` (`gsl_complex a`)

### Square root and exponent functions

- `gsl_complex sqrt` (`gsl_complex a`)
- `gsl_complex sqrt_real` (`double x`)
- `gsl_complex pow` (`gsl_complex a`, `gsl_complex b`)
- `gsl_complex pow_real` (`gsl_complex a`, `double b`)

### Logarithmic and exponential functions

- double **logabs** (gsl\_complex z)
- gsl\_complex **exp** (gsl\_complex a)
- gsl\_complex **log** (gsl\_complex a)
- gsl\_complex **log10** (gsl\_complex a)
- gsl\_complex **log\_b** (gsl\_complex a, gsl\_complex b)

### Trigonometric functions

- gsl\_complex **sin** (gsl\_complex a)
- gsl\_complex **cos** (gsl\_complex a)
- gsl\_complex **tan** (gsl\_complex a)
- gsl\_complex **sec** (gsl\_complex a)
- gsl\_complex **csc** (gsl\_complex a)
- gsl\_complex **cot** (gsl\_complex a)
- gsl\_complex **asin** (gsl\_complex a)
- gsl\_complex **asin\_real** (double a)
- gsl\_complex **acos** (gsl\_complex a)
- gsl\_complex **acos\_real** (double a)
- gsl\_complex **atan** (gsl\_complex a)
- gsl\_complex **asec** (gsl\_complex a)
- gsl\_complex **asec\_real** (double a)
- gsl\_complex **acsc** (gsl\_complex a)
- gsl\_complex **acsc\_real** (double a)
- gsl\_complex **acot** (gsl\_complex a)

### Hyperbolic trigonometric functions

- gsl\_complex **sinh** (gsl\_complex a)
- gsl\_complex **cosh** (gsl\_complex a)
- gsl\_complex **tanh** (gsl\_complex a)
- gsl\_complex **sech** (gsl\_complex a)
- gsl\_complex **csch** (gsl\_complex a)
- gsl\_complex **coth** (gsl\_complex a)
- gsl\_complex **asinh** (gsl\_complex a)
- gsl\_complex **acosh** (gsl\_complex a)
- gsl\_complex **acosh\_real** (double a)
- gsl\_complex **atanh** (gsl\_complex a)
- gsl\_complex **atanh\_real** (double a)
- gsl\_complex **asech** (gsl\_complex a)
- gsl\_complex **acsch** (gsl\_complex a)
- gsl\_complex **acoth** (gsl\_complex a)

## 4.4 `err_hnd.h` File Reference

### 4.4.1 Detailed Description

File for definitions for [err\\_class](#).

Definition in file [err\\_hnd.h](#).

```
#include <iostream>
#include <string>
#include <gsl/gsl_errno.h>
```

### Data Structures

- class [err\\_class](#)  
*The error handler.*

## Defines

- `#define set_err(d, n) o2scl::set_err_fn(d, __FILE__, __LINE__, n);`  
*Set an error.*
- `#define set_err_ret(d, n) do { o2scl::set_err_fn(d, __FILE__, __LINE__, n); return n; } while (0)`  
*Set an error and return the error value.*
- `#define add_err(d, n) o2scl::add_err_fn(d, __FILE__, __LINE__, n);`  
*Set an error and add the information from the last error.*
- `#define add_err_ret(d, n) do { o2scl::add_err_fn(d, __FILE__, __LINE__, n); return n; } while(0)`  
*Set an error, add the information from the last error, and return the error value.*
- `#define err_print(ev)`  
*Print out error information.*
- `#define cerr_print(ev)`  
*Print out error information to `cerr`, do nothing occurred.*
- `#define err_assert(ev)`  
*A version of `assert`, i.e. exit if the error value is non-zero and do nothing otherwise.*
- `#define bool_assert(ev, str)`  
*A version of `assert` for `bool` types. Exit if the argument is false.*

## Enumerations

- `enum {`  
`gsl_success = 0, gsl_failure = -1, gsl_continue = -2, gsl_edom = 1,`  
`gsl_erange = 2, gsl_efault = 3, gsl_einval = 4, gsl_efailed = 5,`  
`gsl_efactor = 6, gsl_esanity = 7, gsl_enomem = 8, gsl_ebadfunc = 9,`  
`gsl_erunaway = 10, gsl_emaxiter = 11, gsl_ezerodiv = 12, gsl_ebadtol = 13,`  
`gsl_etol = 14, gsl_eundrflw = 15, gsl_eovrflw = 16, gsl_eloss = 17,`  
`gsl_eround = 18, gsl_ebadlen = 19, gsl_enotsqr = 20, gsl_esign = 21,`  
`gsl_ediverge = 22, gsl_eunsup = 23, gsl_eunimpl = 24, gsl_ecache = 25,`  
`gsl_etable = 26, gsl_enoproj = 27, gsl_enoproj = 28, gsl_etolf = 29,`  
`gsl_etolx = 30, gsl_etolg = 31, gsl_eof = 32, gsl_nobase = 33,`  
`gsl_notfound = 34, gsl_memtype = 35, gsl_eilenotfound = 36, gsl_index = 37 }`  
*The error definitions from GSL.*

## Functions

- `void set_err_fn (const char *desc, const char *file, int line, int errnum)`  
*Set an error.*
- `void add_err_fn (const char *desc, const char *file, int line, int errnum)`  
*Set an error and add the information from the last error.*
- `void error_update (int &ret, int err)`  
*Desc.*

## Variables

- `err_class * err_hnd`  
*The global error handler pointer.*
- `err_class def_err_hnd`  
*The default error handler.*



## 4.4.2 Define Documentation

### 4.4.2.1 #define cerr\_print(ev)

**Value:**

```
do { if (ev!=0) std::cerr << ev << " " << err_hnd->get_str() << std::endl; \
} while (0)
```

Print out error information to `cerr`, do nothing occurred.

Definition at line 279 of file `err_hnd.h`.

### 4.4.2.2 #define err\_assert(ev)

A version of `assert`, i.e. exit if the error value is non-zero and do nothing otherwise.

#### Todo

Should make this consistent with `assert()` using `NDEBUG`

Definition at line 288 of file `err_hnd.h`.

### 4.4.2.3 #define err\_print(ev)

**Value:**

```
do { if (ev!=0) std::cout << ev << " " << err_hnd->get_str() << std::endl; \
else std::cout << "No error occurred." << std::endl; } while (0)
```

Print out error information.

Definition at line 272 of file `err_hnd.h`.

## 4.4.3 Enumeration Type Documentation

### 4.4.3.1 anonymous enum

The error definitions from GSL.

**Enumerator:**

***gsl\_success*** Success.

***gsl\_failure*** Failure.

***gsl\_continue*** iteration has not converged

***gsl\_edom*** input domain error, e.g. `sqrt(-1)`

***gsl\_erange*** output range error, e.g. `exp(1e100)`

***gsl\_efault*** invalid pointer

***gsl\_einval*** invalid argument supplied by user

***gsl\_efailed*** generic failure

***gsl\_efactor*** factorization failed

***gsl\_esanity*** sanity check failed - shouldn't happen

***gsl\_enomem*** malloc failed

***gsl\_ebadfunc*** problem with user-supplied function

***gsl\_erunaway*** iterative process is out of control  
***gsl\_emaxiter*** exceeded max number of iterations  
***gsl\_ezerodiv*** tried to divide by zero  
***gsl\_ebadtol*** user specified an invalid tolerance  
***gsl\_etol*** failed to reach the specified tolerance  
***gsl\_eundrflw*** underflow  
***gsl\_eovrflw*** overflow  
***gsl\_eloss*** loss of accuracy  
***gsl\_eround*** failed because of roundoff error  
***gsl\_ebadlen*** matrix, vector lengths are not conformant  
***gsl\_enotsqr*** matrix not square  
***gsl\_esing*** apparent singularity detected  
***gsl\_ediverge*** integral or series is divergent  
***gsl\_eunsup*** requested feature is not supported by the hardware  
***gsl\_eunimpl*** requested feature not (yet) implemented  
***gsl\_ecache*** cache limit exceeded  
***gsl\_etable*** [table](#) limit exceeded  
***gsl\_enoprog*** iteration is not making progress toward solution  
***gsl\_enoprogi*** [jacobian](#) evaluations are not improving the solution  
***gsl\_etolf*** cannot reach the specified tolerance in f  
***gsl\_etolx*** cannot reach the specified tolerance in x  
***gsl\_etolg*** cannot reach the specified tolerance in [gradient](#)  
***gsl\_eof*** end of file  
***gsl\_nobase*** a blank method in a base class has been called  
***gsl\_notfound*** Generic "not found" result.  
***gsl\_memtype*** Incorrect type for memory object.  
***gsl\_efilenotfound*** File not found.  
***gsl\_index*** Invalid index for array or matrix.

Definition at line 42 of file `err_hnd.h`.

## 4.5 lib\_settings.h File Reference

### 4.5.1 Detailed Description

File for definitions for [lib\\_settings\\_class](#).

Definition in file [lib\\_settings.h](#).

```
#include <iostream>
```

```
#include <string>
```

### Data Structures

- class [lib\\_settings\\_class](#)  
*A class to manage testing and record success and failure.*

## Variables

- `lib_settings_class lib_settings`  
*The global library settings object.*

## 4.6 minimize.h File Reference

### 4.6.1 Detailed Description

One-dimensional minimization routines.

Definition in file [minimize.h](#).

## Data Structures

- class `minimize`  
*Numerical differentiation base.*

## Functions

- double `constraint` (double `x`, double `center`, double `width`, double `height`)  
*Constrain `x` to be within `width` of the value given by `center`.*
- double `cont_constraint` (double `x`, double `center`, double `width`, double `height`, double `tightness=40.0`, double `exp_arg_limit=50.0`)  
*Constrain `x` to be within `width` of the value given by `center`.*
- double `lower_bound` (double `x`, double `center`, double `width`, double `height`)  
*Constrain `x` to be greater than the value given by `center`.*
- double `cont_lower_bound` (double `x`, double `center`, double `width`, double `height`, double `tightness=40.0`, double `exp_arg_limit=50.0`)  
*Constrain `x` to be greater than the value given by `center`.*

### 4.6.2 Function Documentation

#### 4.6.2.1 `double constraint (double x, double center, double width, double height)` [inline]

Constrain `x` to be within `width` of the value given by `center`.

Defining  $c = \text{center}$ ,  $w = \text{width}$ ,  $h = \text{height}$ , this returns the value  $h(1 + |x - c - w|/w)$  if  $x > c + w$  and  $h(1 + |x - c + w|/w)$  if  $x < c - w$ . The value at  $x = c - w$  or  $x = c + w$  is  $h$  and the value at  $x = c - 2w$  or  $x = c + 2w$  is  $2h$ .

It is important to note that, for large distances of `x` from `center`, this only scales linearly. If you are trying to constrain a function which decreases more than linearly by making `x` far from `center`, then a minimizer will likely ignore this constraint.

Definition at line 237 of file `minimize.h`.

#### 4.6.2.2 `double cont_constraint (double x, double center, double width, double height, double tightness = 40.0, double exp_arg_limit = 50.0)` [inline]

Constrain `x` to be within `width` of the value given by `center`.

Defining  $c = \text{center}$ ,  $w = \text{width}$ ,  $h = \text{height}$ ,  $t = \text{tightness}$ , and  $\ell = \text{exp\_arg\_limit}$ , this returns the value

$$\left(\frac{x - c}{w}\right)^2 \left[1 + e^{t(x - c - w)(c + w - x)/w^2}\right]^{-1}$$

if  $x \geq c$ .

The exponential is handled gracefully by assuming that anything smaller than  $\exp(-\ell)$  is zero. This function is continuous and differentiable. Note that if  $x = c$ , then the function returns zero.

It is important to note that, for large distances of  $x$  from `center`, this only scales linearly. If you are trying to constrain a function which decreases more than linearly by making  $x$  far from `center`, then a minimizer will likely ignore this constraint.

Definition at line 272 of file `minimize.h`.

**4.6.2.3 `double cont_lower_bound (double x, double center, double width, double height, double tightness = 40.0, double exp_arg_limit = 50.0) [inline]`**

Constrain  $x$  to be greater than the value given by `center`.

Defining  $c = \text{center}$ ,  $w = \text{width}$ ,  $h = \text{height}$ ,  $t = \text{tightness}$ , and  $\ell = \text{exp\_arg\_limit}$ , this returns  $h(c - x + w)/(1 + \exp(t(x - c)/w))$  and has the advantage of being a continuous and differentiable function. The exponential is handled gracefully by assuming that anything smaller than  $\exp(-\ell)$  is zero

It is important to note that, for large distances of  $x$  from `center`, this only scales linearly. If you are trying to constrain a function which decreases more than linearly by making  $x$  far from `center`, then the constraint will be essentially ignored.

Definition at line 324 of file `minimize.h`.

**4.6.2.4 `double lower_bound (double x, double center, double width, double height) [inline]`**

Constrain  $x$  to be greater than the value given by `center`.

Defining  $c = \text{center}$ ,  $w = \text{width}$ ,  $h = \text{height}$ , this returns  $h(1 + |x - c|/w)$  if  $x < c$  and zero otherwise. The value at  $x = c$  is  $h$ , while the value at  $x = c - w$  is  $2h$ .

It is important to note that, for large distances of  $x$  from `center`, this only scales linearly. If you are trying to constrain a function which decreases more than linearly by making  $x$  far from `center`, then a minimizer will likely ignore this constraint.

Definition at line 300 of file `minimize.h`.

## 4.7 misc.h File Reference

### 4.7.1 Detailed Description

Miscellaneous functions.

Definition in file [misc.h](#).

```
#include <iostream>
#include <cmath>
#include <string>
#include <fstream>
#include <sstream>
#include <o2scl/err_hnd.h>
#include <o2scl/lib_settings.h>
#include <gsl/gsl_ieee_utils.h>
```

### Data Structures

- struct [string\\_comp](#)  
*Naive string comparison.*
- class [gen\\_test\\_number](#)

*Generate number sequence for testing.*

## Functions

- double [fermi\\_function](#) (double *E*, double *mu*, double *T*, double *limit*=40.0)  
*Calculate a Fermi-Dirac distribution function safely.*
- void [screenify](#) (const std::string \**in\_cols*, int *nin*, std::string \*&*outc*, int &*nout*, int *max\_size*=80)  
*Reformat the columns for output of width size.*
- int [count\\_words](#) (std::string *str*)  
*Count the number of words in the string str.*
- std::string [binary\\_to\\_hex](#) (std::string *s*)  
*Take a string of binary quads and compress them to hexadecimal digits.*
- template<class type>  
type \*\* [new\\_2d\\_array](#) (size\_t *nr*, size\_t *nc*)  
*Create a new C-style 2-dimensional array.*
- template<class type>  
int [delete\\_2d\\_array](#) (type \*\**t*, size\_t *nr*)  
*Create a new C-style 2-dimensional array.*

### 4.7.2 Function Documentation

#### 4.7.2.1 std::string binary\_to\_hex (std::string *s*)

Take a string of binary quads and compress them to hexadecimal digits.

This function proceeds from left to right, ignoring parts of the string that do not consist of sequences of four '1's or '0's.

#### 4.7.2.2 int count\_words (std::string *str*)

Count the number of words in the string *str*.

Words are defined as groups of characters separated by whitespace, where whitespace is any combination of adjacent spaces, tabs, carriage returns, etc. On most systems, whitespace is usually defined as any character corresponding to the integers 9 (horizontal tab), 10 (line feed), 11 (vertical tab), 12 (form feed), 13 (carriage return), and 32 (space bar). The test program `misc_ts` enumerates the characters between 0 and 255 (inclusive) that count as whitespace for this purpose.

Note that this function is used in [text\\_in\\_file::string\\_in](#) to perform string input.

#### 4.7.2.3 double fermi\_function (double *E*, double *mu*, double *T*, double *limit* = 40.0)

Calculate a Fermi-Dirac distribution function safely.

$$[1 + \exp(E/T - \mu/T)]^{-1}$$

This calculates a Fermi-Dirac distribution function guaranteeing that numbers larger than  $\exp(\text{limit})$  and smaller than  $\exp(-\text{limit})$  will be avoided. The default value of *limit* ensures accuracy to within 1 part in  $10^{17}$  compared to the maximum of the distribution (which is unity).

Note that this function may return Inf or NAN if *limit* is too large, depending on the machine precision.

#### 4.7.2.4 void screenify (const std::string \**in\_cols*, int *nin*, std::string \*&*outc*, int &*nout*, int *max\_size* = 80)

Reformat the columns for output of width *size*.

Given a string array *in\_cols* of size *nin*, [screenify\(\)](#) reformats the array into columns creating a new string array *outc* with size *nout*.

For example, for an array of 10 strings

```

test1
test_of_string2
test_of_string3
test_of_string4
test5
test_of_string6
test_of_string7
test_of_string8
test_of_string9
test_of_string10

```

`screenify()` will create an array of 3 new strings:

```

test1          test_of_string4  test_of_string7  test_of_string10
test_of_string2 test5          test_of_string8
test_of_string3 test_of_string6  test_of_string9

```

The string array given in `outc` must be deleted with `delete[]` after usage.

If the value of `max_size` is less than the length of the longest input string (plus one for a space character), then the output strings may have a larger length than `max_size`.

### Todo

Consider making this into a class so that the memory for the output columns is automatically handled.

## 4.8 `omatrix_cx_tlate.h` File Reference

### 4.8.1 Detailed Description

File for definitions of complex matrices.

Definition in file `omatrix_cx_tlate.h`.

```

#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <o2scl/err_hnd.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_complex.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/ovector_cx_tlate.h>

```

### Data Structures

- class `omatrix_cx_view_tlate`  
*A matrix view of double-precision numbers.*
- class `omatrix_cx_tlate`  
*A matrix of double-precision numbers.*
- class `omatrix_cx_row_tlate`

*Create a vector from a row of a matrix.*

- class [omatrix\\_cx\\_const\\_row\\_tlate](#)  
*Create a vector from a row of a matrix.*
- class [omatrix\\_cx\\_col\\_tlate](#)  
*Create a vector from a column of a matrix.*
- class [omatrix\\_cx\\_const\\_col\\_tlate](#)  
*Create a vector from a column of a matrix.*

## Typedefs

- typedef [omatrix\\_cx\\_tlate](#)< double, gsl\_matrix\_complex, gsl\_block\_complex, gsl\_complex > [omatrix\\_cx](#)  
*omatrix\_cx typedef*
- typedef [omatrix\\_cx\\_view\\_tlate](#)< double, gsl\_matrix\_complex, gsl\_block\_complex, gsl\_complex > [omatrix\\_cx\\_view](#)  
*omatrix\_cx\_view typedef*
- typedef [omatrix\\_cx\\_row\\_tlate](#)< double, gsl\_matrix\_complex, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [omatrix\\_cx\\_row](#)  
*omatrix\_cx\_row typedef*
- typedef [omatrix\\_cx\\_col\\_tlate](#)< double, gsl\_matrix\_complex, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [omatrix\\_cx\\_col](#)  
*omatrix\_cx\_col typedef*
- typedef [omatrix\\_cx\\_const\\_row\\_tlate](#)< double, gsl\_matrix\_complex, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [omatrix\\_cx\\_const\\_row](#)  
*omatrix\_cx\_const\_row typedef*
- typedef [omatrix\\_cx\\_const\\_col\\_tlate](#)< double, gsl\_matrix\_complex, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [omatrix\\_cx\\_const\\_col](#)  
*omatrix\_cx\_const\_col typedef*

## 4.9 omatrix\_tlate.h File Reference

### 4.9.1 Detailed Description

File for definitions of matrices.

Definition in file [omatrix\\_tlate.h](#).

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_ieee_utils.h>
#include <o2scl/err_hnd.h>
#include <o2scl/ovector_tlate.h>
```

## Data Structures

- class [omatrix\\_view\\_tlate](#)  
*A matrix view of double-precision numbers.*
- class [omatrix\\_tlate](#)  
*A matrix of double-precision numbers.*
- class [omatrix\\_row\\_tlate](#)

*Create a vector from a row of a matrix.*

- class `omatrix_const_row_tlate`  
*Create a const vector from a row of a matrix.*
- class `omatrix_col_tlate`  
*Create a vector from a column of a matrix.*
- class `omatrix_const_col_tlate`  
*Create a const vector from a column of a matrix.*
- class `omatrix_diag_tlate`  
*Create a vector from the main diagonal.*
- class `omatrix_alloc`  
*A simple class to provide an `allocate()` function for `omatrix`.*
- class `ofmatrix`  
*A matrix where the memory allocation is performed in the constructor.*

## Typedefs

- typedef `omatrix_tlate`< double, gsl\_matrix, gsl\_vector, gsl\_block > `omatrix`  
*omatrix typedef*
- typedef `omatrix_view_tlate`< double, gsl\_matrix, gsl\_block > `omatrix_view`  
*omatrix\_view typedef*
- typedef `omatrix_row_tlate`< double, gsl\_matrix, gsl\_vector, gsl\_block > `omatrix_row`  
*omatrix\_row typedef*
- typedef `omatrix_col_tlate`< double, gsl\_matrix, gsl\_vector, gsl\_block > `omatrix_col`  
*omatrix\_col typedef*
- typedef `omatrix_const_row_tlate`< double, gsl\_matrix, gsl\_vector, gsl\_block > `omatrix_const_row`  
*omatrix\_const\_row typedef*
- typedef `omatrix_const_col_tlate`< double, gsl\_matrix, gsl\_vector, gsl\_block > `omatrix_const_col`  
*omatrix\_const\_col typedef*
- typedef `omatrix_diag_tlate`< double, gsl\_matrix, gsl\_vector, gsl\_block > `omatrix_diag`  
*omatrix\_diag typedef*
- typedef `omatrix_tlate`< int, gsl\_matrix\_int, gsl\_vector\_int, gsl\_block\_int > `omatrix_int`  
*omatrix\_int typedef*
- typedef `omatrix_view_tlate`< int, gsl\_matrix\_int, gsl\_block\_int > `omatrix_int_view`  
*omatrix\_int\_view typedef*
- typedef `omatrix_row_tlate`< int, gsl\_matrix\_int, gsl\_vector\_int, gsl\_block\_int > `omatrix_int_row`  
*omatrix\_int\_row typedef*
- typedef `omatrix_col_tlate`< int, gsl\_matrix\_int, gsl\_vector\_int, gsl\_block\_int > `omatrix_int_col`  
*omatrix\_int\_col typedef*
- typedef `omatrix_const_row_tlate`< int, gsl\_matrix\_int, gsl\_vector\_int, gsl\_block\_int > `omatrix_int_const_row`  
*omatrix\_int\_const\_row typedef*
- typedef `omatrix_const_col_tlate`< int, gsl\_matrix\_int, gsl\_vector\_int, gsl\_block\_int > `omatrix_int_const_col`  
*omatrix\_int\_const\_col typedef*
- typedef `omatrix_diag_tlate`< int, gsl\_matrix\_int, gsl\_vector\_int, gsl\_block\_int > `omatrix_int_diag`  
*omatrix\_int\_diag typedef*

## Functions

- template<class data\_t, class parent\_t, class block\_t>  
std::ostream & `operator<<` (std::ostream &os, const `omatrix_view_tlate`< data\_t, parent\_t, block\_t > &v)  
*A operator for naive matrix output.*



## 4.9.2 Function Documentation

### 4.9.2.1 `std::ostream& operator<< (std::ostream & os, const omatrix_view_tlate< data_t, parent_t, block_t > & v)` [inline]

A operator for naive matrix output.

This outputs all of the matrix elements. Each row is output with an newline character at the end of each row. Positive values are preceded by an extra space. A 2x2 example:

```
-3.751935e-05 -6.785864e-04
-6.785864e-04  1.631984e-02
```

The function `gsl_ieee_double_to_rep()` is used to determine the sign of a number, so that "-0.0" as distinct from "+0.0" is handled correctly.

#### Todo

Maybe remove this function, as it's superceded by [matrix\\_out\(\)](#)?

#### Todo

This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Definition at line 840 of file `omatrix_tlate.h`.

## 4.10 `ovector_cx_tlate.h` File Reference

### 4.10.1 Detailed Description

File for definitions of complex vectors.

Definition in file [ovector\\_cx\\_tlate.h](#).

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <complex>
#include <o2scl/err_hnd.h>
#include <o2scl/ovector_tlate.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_complex.h>
```

#### Data Structures

- class [ovector\\_cx\\_view\\_tlate](#)  
A vector view of double-precision numbers.
- class [ovector\\_cx\\_tlate](#)

*A vector of double-precision numbers.*

- class [ovector\\_cx\\_array\\_tlate](#)  
*Create a vector from an array.*
- class [ovector\\_cx\\_array\\_stride\\_tlate](#)  
*Create a vector from an array with a stride.*
- class [ovector\\_cx\\_subvector\\_tlate](#)  
*Create a vector from a subvector of another.*
- class [ovector\\_cx\\_const\\_array\\_tlate](#)  
*Create a vector from an array.*
- class [ovector\\_cx\\_const\\_array\\_stride\\_tlate](#)  
*Create a vector from an array\_stride.*
- class [ovector\\_cx\\_const\\_subvector\\_tlate](#)  
*Create a vector from a subvector of another.*
- class [ovector\\_cx\\_real\\_tlate](#)  
*Create a real vector from the real parts of a complex vector.*
- class [ovector\\_cx\\_imag\\_tlate](#)  
*Create a imaginary vector from the imaginary parts of a complex vector.*
- class [ovector\\_cx](#)  
*A vector where the memory allocation is performed in the constructor.*

## Typedefs

- typedef [ovector\\_cx\\_tlate](#)< double, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [ovector\\_cx](#)  
*ovector\_cx typedef*
- typedef [ovector\\_cx\\_view\\_tlate](#)< double, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [ovector\\_cx\\_view](#)  
*ovector\_cx\_view typedef*
- typedef [ovector\\_cx\\_array\\_tlate](#)< double, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [ovector\\_cx\\_array](#)  
*ovector\_cx\_array typedef*
- typedef [ovector\\_cx\\_array\\_stride\\_tlate](#)< double, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [ovector\\_cx\\_array\\_stride](#)  
*ovector\_cx\_array\_stride typedef*
- typedef [ovector\\_cx\\_subvector\\_tlate](#)< double, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [ovector\\_cx\\_subvector](#)  
*ovector\_cx\_subvector typedef*
- typedef [ovector\\_cx\\_const\\_array\\_tlate](#)< double, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [ovector\\_cx\\_const\\_array](#)  
*ovector\_cx\_const\_array typedef*
- typedef [ovector\\_cx\\_const\\_array\\_stride\\_tlate](#)< double, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [ovector\\_cx\\_const\\_array\\_stride](#)  
*ovector\_cx\_const\_array\_stride typedef*
- typedef [ovector\\_cx\\_const\\_subvector\\_tlate](#)< double, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [ovector\\_cx\\_const\\_subvector](#)  
*ovector\_cx\_const\_subvector typedef*
- typedef [ovector\\_cx\\_real\\_tlate](#)< double, gsl\_vector, gsl\_block, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [ovector\\_cx\\_real](#)  
*ovector\_cx\_real typedef*
- typedef [ovector\\_cx\\_imag\\_tlate](#)< double, gsl\_vector, gsl\_block, gsl\_vector\_complex, gsl\_block\_complex, gsl\_complex > [ovector\\_cx\\_imag](#)  
*ovector\_cx\_imag typedef*

## Functions

- gsl\_complex [complex\\_to\\_gsl](#) (std::complex< double > &d)  
*Convert a complex number to GSL form.*
- std::complex< double > [gsl\\_to\\_complex](#) (gsl\_complex &g)  
*Convert a complex number to STL form.*

- `template<class data_t, class vparent_t, class block_t, class complex_t>`  
`ovector_cx_tlate< data_t, vparent_t, block_t, complex_t > conjugate (ovector_cx_tlate< data_t, vparent_t, block_t, complex_t > &v)`  
*Conjugate a vector.*
- `template<class data_t, class vparent_t, class block_t, class complex_t>`  
`std::ostream & operator<< (std::ostream &os, const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &v)`  
*A operator for naive vector output.*

## 4.10.2 Function Documentation

### 4.10.2.1 `std::ostream& operator<< (std::ostream & os, const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &v)` [inline]

A operator for naive vector output.

This outputs all of the vector elements in the form (r,i). All of these are separated by one space character, though no trailing space or `endl` is sent to the output.

Definition at line 1066 of file `ovector_cx_tlate.h`.

## 4.11 ovector\_tlate.h File Reference

### 4.11.1 Detailed Description

File for definitions of vectors.

Definition in file `ovector_tlate.h`.

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <gsl/gsl_vector.h>
#include <o2scl/err_hnd.h>
#include <o2scl/string_conv.h>
#include <o2scl/uvector_tlate.h>
#include <o2scl/array.h>
```

### Data Structures

- class `ovector_view_tlate`  
*A vector view with finite stride.*
- class `ovector_tlate`  
*A vector with finite stride.*
- class `ovector_array_tlate`  
*Create a vector from an array.*
- class `ovector_array_stride_tlate`  
*Create a vector from an array with a stride.*
- class `ovector_subvector_tlate`  
*Create a vector from a subvector of another.*

- class [ovector\\_const\\_array\\_tlate](#)  
*Create a const vector from an array.*
- class [ovector\\_const\\_array\\_stride\\_tlate](#)  
*Create a const vector from an array with a stride.*
- class [ovector\\_const\\_subvector\\_tlate](#)  
*Create a const vector from a subvector of another vector.*
- class [ovector\\_reverse\\_tlate](#)  
*Reversed view of a vector.*
- class [ovector\\_const\\_reverse\\_tlate](#)  
*Reversed view of a vector.*
- class [ovector\\_subvector\\_reverse\\_tlate](#)  
*Reversed view of a subvector.*
- class [ovector\\_const\\_subvector\\_reverse\\_tlate](#)  
*Reversed view of a const subvector.*
- class [ovector\\_alloc](#)  
*A simple class to provide an `allocate()` function for `ovector`.*
- class [ovector\\_int\\_alloc](#)  
*A simple class to provide an `allocate()` function for `ovector_int`.*
- class [ovector](#)  
*A vector where the memory allocation is performed in the constructor.*

## Typedefs

- typedef [ovector\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector](#)  
*ovector typedef*
- typedef [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > [ovector\\_view](#)  
*ovector\_view typedef*
- typedef [ovector\\_array\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector\\_array](#)  
*ovector\_array typedef*
- typedef [ovector\\_array\\_stride\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector\\_array\\_stride](#)  
*ovector\_array\_stride typedef*
- typedef [ovector\\_subvector\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector\\_subvector](#)  
*ovector\_subvector typedef*
- typedef [ovector\\_const\\_array\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector\\_const\\_array](#)  
*ovector\_const\_array typedef*
- typedef [ovector\\_const\\_array\\_stride\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector\\_const\\_array\\_stride](#)  
*ovector\_const\_array\_stride typedef*
- typedef [ovector\\_const\\_subvector\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector\\_const\\_subvector](#)  
*ovector\_const\_subvector typedef*
- typedef [ovector\\_reverse\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector\\_reverse](#)  
*ovector\_reverse typedef*
- typedef [ovector\\_const\\_reverse\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector\\_const\\_reverse](#)  
*ovector\_const\_reverse typedef*
- typedef [ovector\\_subvector\\_reverse\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector\\_subvector\\_reverse](#)  
*ovector\_subvector\_reverse typedef*
- typedef [ovector\\_const\\_subvector\\_reverse\\_tlate](#)< double, gsl\_vector, gsl\_block > [ovector\\_const\\_subvector\\_reverse](#)  
*ovector\_const\_subvector\_reverse typedef*
- typedef [ovector\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int](#)  
*ovector\_int typedef*
- typedef [ovector\\_view\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_view](#)  
*ovector\_int\_view typedef*
- typedef [ovector\\_array\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_array](#)  
*ovector\_int\_array typedef*
- typedef [ovector\\_array\\_stride\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_array\\_stride](#)  
*ovector\_int\_array\_stride typedef*
- typedef [ovector\\_subvector\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_subvector](#)  
*ovector\_int\_subvector typedef*

- typedef [ovector\\_const\\_array\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_const\\_array](#)  
*ovector\_int\_const\_array typedef*
- typedef [ovector\\_const\\_array\\_stride\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_const\\_array\\_stride](#)  
*ovector\_int\_const\_array\_stride typedef*
- typedef [ovector\\_const\\_subvector\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_const\\_subvector](#)  
*ovector\_int\_const\_subvector typedef*
- typedef [ovector\\_reverse\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_reverse](#)  
*ovector\_int\_reverse typedef*
- typedef [ovector\\_const\\_reverse\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_const\\_reverse](#)  
*ovector\_int\_const\_reverse typedef*
- typedef [ovector\\_subvector\\_reverse\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_subvector\\_reverse](#)  
*ovector\_int\_subvector\_reverse typedef*
- typedef [ovector\\_const\\_subvector\\_reverse\\_tlate](#)< int, gsl\_vector\_int, gsl\_block\_int > [ovector\\_int\\_const\\_subvector\\_reverse](#)  
*ovector\_int\_const\_subvector\_reverse typedef*

## Functions

- template<class data\_t, class vparent\_t, class block\_t>  
std::ostream & [operator<<](#) (std::ostream & os, const [ovector\\_view\\_tlate](#)< data\_t, vparent\_t, block\_t > &v)  
*A operator for naive vector output.*

### 4.11.2 Function Documentation

**4.11.2.1 std::ostream& operator<< (std::ostream & os, const ovector\_view\_tlate< data\_t, vparent\_t, block\_t > & v)**  
[inline]

A operator for naive vector output.

This outputs all of the vector elements. All of these are separated by one space character, though no trailing space or endl is sent to the output.

Definition at line 1835 of file ovector\_tlate.h.

## 4.12 poly.h File Reference

### 4.12.1 Detailed Description

Classes for solving polynomials.

#### Warning:

We should be careful about using pow() in functions using complex<double> since pow(((complex<double>)0.0),3.0) returns (nan,nan). Instead, we should use pow(((complex<double>)0.0),3) which takes an integer for the second argument. The sqrt() function, always succeeds i.e. sqrt(((complex<double>)0.0))=0.0

One has to be careful about using e.g. pow(a,1.0/3.0) for complex a since if Re(a)<0 and Im(a)==0 then the function returns NaN.

Definition in file [poly.h](#).

```
#include <iostream>
#include <complex>
#include <gsl/gsl_math.h>
#include <gsl/gsl_complex_math.h>
#include <gsl/gsl_complex.h>
```

```
#include <gsl/gsl_poly.h>
#include <o2scl/constants.h>
#include <o2scl/err_hnd.h>
```

## Data Structures

- class [quadratic\\_real](#)  
*Solve a quadratic polynomial with real coefficients and real roots.*
- class [quadratic\\_real\\_coeff](#)  
*Solve a quadratic polynomial with real coefficients and complex roots.*
- class [quadratic\\_complex](#)  
*Solve a quadratic polynomial with complex coefficients and complex roots.*
- class [cubic\\_real](#)  
*Solve a cubic polynomial with real coefficients and real roots.*
- class [cubic\\_real\\_coeff](#)  
*Solve a cubic polynomial with real coefficients and complex roots.*
- class [cubic\\_complex](#)  
*Solve a cubic polynomial with complex coefficients and complex roots.*
- class [quartic\\_real](#)  
*Solve a quartic polynomial with real coefficients and real roots.*
- class [quartic\\_real\\_coeff](#)  
*Solve a quartic polynomial with real coefficients and complex roots.*
- class [quartic\\_complex](#)  
*Solve a quartic polynomial with complex coefficients and complex roots.*
- class [poly\\_real\\_coeff](#)  
*Base class for solving a general polynomial with real coefficients and complex roots.*
- class [poly\\_complex](#)  
*Base class for solving a general polynomial with complex coefficients.*
- class [cern\\_cubic\\_real\\_coeff](#)  
*Solve a cubic with real coefficients and complex roots (CERNLIB).*
- class [cern\\_quartic\\_real\\_coeff](#)  
*Solve a quartic with real coefficients and complex roots (CERNLIB).*
- class [gsl\\_quadratic\\_real\\_coeff](#)  
*Solve a quadratic with real coefficients and complex roots (GSL).*
- class [gsl\\_cubic\\_real\\_coeff](#)  
*Solve a cubic with real coefficients and complex roots (GSL).*
- class [gsl\\_quartic\\_real](#)  
*Solve a quartic with real coefficients and real roots (GSL).*
- class [gsl\\_quartic\\_real2](#)  
*Solve a quartic with real coefficients and real roots (GSL).*
- class [gsl\\_poly\\_real\\_coeff](#)  
*Solve a general polynomial with real coefficients (GSL).*
- class [quadratic\\_std\\_complex](#)  
*Solve a quadratic with complex coefficients and complex roots.*
- class [cubic\\_std\\_complex](#)  
*Solve a cubic with complex coefficients and complex roots.*
- class [naive\\_quartic\\_real](#)  
*Solve a quartic with real coefficients and real roots.*
- class [naive\\_quartic\\_complex](#)  
*Solve a quartic with complex coefficients and complex roots.*

## 4.13 string\_conv.h File Reference

### 4.13.1 Detailed Description

Various string conversion functions.

---

Definition in file [string\\_conv.h](#).

```
#include <iostream>
#include <cmath>
#include <string>
#include <fstream>
#include <sstream>
#include <o2scl/err_hnd.h>
#include <o2scl/lib_settings.h>
#include <gsl/gsl_ieee_utils.h>
```

## Functions

- `std::string ptos (void *p)`  
*Convert a pointer to a string.*
- `std::string itos (int x)`  
*Convert an integer to a string.*
- `std::string dtos (double x, int prec=6, bool auto_prec=false)`  
*Convert a double to a string.*
- `size_t size_of_exponent (double x)`  
*Returns the number of characters required to display the exponent of x in scientific mode.*
- `std::string dtos (double x, std::ostream &format)`  
*Convert a double to a string using a specified format.*
- `int stoi (std::string s)`  
*Convert a string to an integer.*
- `double stod (std::string s)`  
*Convert a string to a double.*
- `std::string double_to_latex (double x, int sigfigs=5, int ex_min=-2, int ex_max=3)`  
*Convert a double to a Latex-like string.*
- `std::string double_to_html (double x, int sigfigs=5, int ex_min=-2, int ex_max=3)`  
*Convert a double to a HTML-like string.*
- `std::string double_to_ieee_string (double *x)`  
*Convert a double to a string containing IEEE representation.*

### 4.13.2 Function Documentation

#### 4.13.2.1 `std::string double_to_html (double x, int sigfigs = 5, int ex_min = -2, int ex_max = 3)`

Convert a double to a HTML-like string.

This uses `&times;`; and `&sup;` to convert a double to a string for use on the web.

The parameter `sigfigs` is the number of significant figures to give in the string. The parameters `ex_min` and `ex_max` represent the base-10 logarithm of the smallest and largest numbers to be represented without exponential notation. The number zero is always converted to "0". Superscripts are implemented using (can't use greater than size in documentation)

Note that this function does not warn the user if the number of significant figures requested is larger than the machine precision.

#### 4.13.2.2 `std::string double_to_ieee_string (double *x)`

Convert a double to a string containing IEEE representation.

Modeled after the GSL function `gsl_ieee_fprintf_double()`, but converts to a string instead of a FILE \*.

**4.13.2.3 `std::string double_to_latex (double x, int sigfigs = 5, int ex_min = -2, int ex_max = 3)`**

Convert a double to a Latex-like string.

The parameter `sigfigs` is the number of significant figures to give in the string. The parameters `ex_min` and `ex_max` represent the base-10 logarithm of the smallest and largest numbers to be represented without the string

```
$\times 10^{\mathrm{ex}}$
```

where `ex` is the relevant exponent. The number zero is always converted to "0".

Note that this function does not warn the user if the number of significant figures requested is larger than the machine precision.

**Todo**

Fix to ensure final zeros are printed properly if requested

**4.13.2.4 `std::string dtos (double x, std::ostream & format)`**

Convert a double to a string using a specified format.

**Todo**

Add error checking to this function using `if (strout << x)`

**4.13.2.5 `std::string ptos (void * p)`**

Convert a pointer to a string.

This uses an `ostream` to convert a pointer to a string and is architecture-dependent.

**4.13.2.6 `size_t size_of_exponent (double x)`**

Returns the number of characters required to display the exponent of `x` in scientific mode.

This usually returns 2 or 3, depending on whether or not the absolute magnitude of the exponent is greater than 100.

**4.13.2.7 `double stod (std::string s)`**

Convert a string to a double.

If this function fails it will call `set_err()` and return zero.

**4.13.2.8 `int stoi (std::string s)`**

Convert a string to an integer.

If this function fails it will call `set_err()` and return zero.

**4.14 `umatrix_cx_tlate.h` File Reference****4.14.1 Detailed Description**

File for definitions of matrices.

Definition in file `umatrix_cx_tlate.h`.

---



```

#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_ieee_utils.h>
#include <o2scl/err_hnd.h>
#include <o2scl/uvectord_tlate.h>
#include <o2scl/uvectord_cx_tlate.h>

```

## Data Structures

- class [umatrix\\_cx\\_view\\_tlate](#)  
*A matrix view of complex numbers.*
- class [umatrix\\_cx\\_tlate](#)  
*A matrix of double-precision numbers.*
- class [umatrix\\_cx\\_row\\_tlate](#)  
*Create a vector from a row of a matrix.*
- class [umatrix\\_cx\\_const\\_row\\_tlate](#)  
*Create a const vector from a row of a matrix.*
- class [umatrix\\_cx\\_alloc](#)  
*A simple class to provide an `allocate()` function for `umatrix_cx`.*
- class [ufmatrix\\_cx](#)  
*A matrix where the memory allocation is performed in the constructor.*

## Typedefs

- typedef [umatrix\\_cx\\_tlate](#)< double, gsl\_complex > [umatrix\\_cx](#)  
*umatrix\_cx typedef*
- typedef [umatrix\\_cx\\_view\\_tlate](#)< double, gsl\_complex > [umatrix\\_cx\\_view](#)  
*umatrix\_cx\_view typedef*
- typedef [umatrix\\_cx\\_row\\_tlate](#)< double, gsl\_complex > [umatrix\\_cx\\_row](#)  
*umatrix\_cx\_row typedef*
- typedef [umatrix\\_cx\\_const\\_row\\_tlate](#)< double, gsl\_complex > [umatrix\\_cx\\_const\\_row](#)  
*umatrix\_cx\_const\_row typedef*

## Functions

- template<class data\_t, class complex\_t>  
std::ostream & [operator<<](#) (std::ostream &os, const [umatrix\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > &v)  
*A operator for naive matrix output.*

### 4.14.2 Function Documentation

**4.14.2.1** `std::ostream& operator<< (std::ostream & os, const umatrix_cx_view_tlate< data_t, complex_t > & v)`  
[inline]

A operator for naive matrix output.

This outputs all of the matrix elements. Each row is output with an newline character at the end of each row. Positive values are preceded by an extra space. A 2x2 example:

```
-3.751935e-05 -6.785864e-04
-6.785864e-04  1.631984e-02
```

The function `gsl_ieee_double_to_rep()` is used to determine the sign of a number, so that "-0.0" as distinct from "+0.0" is handled correctly.

### Todo

This assumes that scientific mode is on and `showpos` is off. It'd be nice to fix this.

Definition at line 657 of file `umatrix_cx_tlate.h`.

## 4.15 `umatrix_tlate.h` File Reference

### 4.15.1 Detailed Description

File for definitions of matrices.

Definition in file `umatrix_tlate.h`.

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_ieee_utils.h>
#include <o2scl/err_hnd.h>
#include <o2scl/uvector_tlate.h>
```

### Data Structures

- class `umatrix_view_tlate`  
*A matrix view of double-precision numbers.*
- class `umatrix_tlate`  
*A matrix of double-precision numbers.*
- class `umatrix_row_tlate`  
*Create a vector from a row of a matrix.*
- class `umatrix_const_row_tlate`  
*Create a const vector from a row of a matrix.*
- class `umatrix_alloc`  
*A simple class to provide an `allocate()` function for `umatrix`.*
- class `ufmatrix`  
*A matrix where the memory allocation is performed in the constructor.*

### Typedefs

- typedef `umatrix_tlate< double > umatrix`

*umatrix typedef*

- typedef `umatrix_view_tlate< double > umatrix_view`  
*umatrix\_view typedef*
- typedef `umatrix_row_tlate< double > umatrix_row`  
*umatrix\_row typedef*
- typedef `umatrix_const_row_tlate< double > umatrix_const_row`  
*umatrix\_const\_row typedef*
- typedef `umatrix_tlate< int > umatrix_int`  
*umatrix\_int typedef*
- typedef `umatrix_view_tlate< int > umatrix_int_view`  
*umatrix\_int\_view typedef*
- typedef `umatrix_row_tlate< int > umatrix_int_row`  
*umatrix\_int\_row typedef*
- typedef `umatrix_const_row_tlate< int > umatrix_int_const_row`  
*umatrix\_int\_const\_row typedef*

## Functions

- template<class data\_t>  
std::ostream & `operator<<` (std::ostream &os, const `umatrix_view_tlate< data_t > &v`)  
*A operator for naive matrix output.*

### 4.15.2 Function Documentation

#### 4.15.2.1 `std::ostream& operator<<` (std::ostream &os, const `umatrix_view_tlate< data_t > &v`) [inline]

A operator for naive matrix output.

This outputs all of the matrix elements. Each row is output with an newline character at the end of each row. Positive values are preceded by an extra space. A 2x2 example:

```
-3.751935e-05 -6.785864e-04
-6.785864e-04  1.631984e-02
```

The function `gsl_ieee_double_to_rep()` is used to determine the sign of a number, so that "-0.0" as distinct from "+0.0" is handled correctly.

## Todo

This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Definition at line 646 of file `umatrix_tlate.h`.

## 4.16 `uvector_cx_tlate.h` File Reference

### 4.16.1 Detailed Description

File for definitions of complex unit-stride vectors.

Definition in file `uvector_cx_tlate.h`.

```
#include <iostream>
#include <cstdlib>
#include <string>
```

```
#include <fstream>
#include <sstream>
#include <vector>
#include <o2scl/err_hnd.h>
#include <gsl/gsl_vector.h>
```

## Data Structures

- class [uvector\\_cx\\_view\\_tlate](#)  
*A vector view of complex numbers with unit stride.*
- class [uvector\\_cx\\_tlate](#)  
*A vector of double-precision numbers with unit stride.*
- class [uvector\\_cx\\_array\\_tlate](#)  
*Create a vector from an array.*
- class [uvector\\_cx\\_subvector\\_tlate](#)  
*Create a vector from a subvector of another.*
- class [uvector\\_cx\\_const\\_array\\_tlate](#)  
*Create a vector from an array.*
- class [uvector\\_cx\\_const\\_subvector\\_tlate](#)  
*Create a vector from a subvector of another.*

## Typedefs

- typedef [uvector\\_cx\\_tlate](#)< double, gsl\_complex > [uvector\\_cx](#)  
*uvector\_cx typedef*
- typedef [uvector\\_cx\\_view\\_tlate](#)< double, gsl\_complex > [uvector\\_cx\\_view](#)  
*uvector\_cx\_view typedef*
- typedef [uvector\\_cx\\_array\\_tlate](#)< double, gsl\_complex > [uvector\\_cx\\_array](#)  
*uvector\_cx\_array typedef*
- typedef [uvector\\_cx\\_subvector\\_tlate](#)< double, gsl\_complex > [uvector\\_cx\\_subvector](#)  
*uvector\_cx\_subvector typedef*
- typedef [uvector\\_cx\\_const\\_array\\_tlate](#)< double, gsl\_complex > [uvector\\_cx\\_const\\_array](#)  
*uvector\_cx\_const\_array typedef*
- typedef [uvector\\_cx\\_const\\_subvector\\_tlate](#)< double, gsl\_complex > [uvector\\_cx\\_const\\_subvector](#)  
*uvector\_cx\_const\_subvector typedef*

## Functions

- template<class data\_t, class complex\_t>  
std::ostream & [operator<<](#) (std::ostream &os, const [uvector\\_cx\\_view\\_tlate](#)< data\_t, complex\_t > &v)  
*A operator for naive vector output.*

### 4.16.2 Function Documentation

#### 4.16.2.1 `std::ostream& operator<< (std::ostream & os, const uvector_cx_view_tlate< data_t, complex_t > & v)` [inline]

A operator for naive vector output.

This outputs all of the vector elements. All of these are separated by one space character, though no trailing space or `endl` is sent to the output.

Definition at line 664 of file `uvector_cx_tlate.h`.

## 4.17 `uvector_tlate.h` File Reference

### 4.17.1 Detailed Description

File for definitions of unit-stride vectors.

Definition in file `uvector_tlate.h`.

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <o2scl/err_hnd.h>
#include <o2scl/string_conv.h>
#include <gsl/gsl_vector.h>
```

### Data Structures

- class `uvector_view_tlate`  
*A vector view with unit stride.*
- class `uvector_tlate`  
*A vector with unit stride.*
- class `uvector_array_tlate`  
*Create a vector from an array.*
- class `uvector_subvector_tlate`  
*Create a vector from a subvector of another.*
- class `uvector_const_array_tlate`  
*Create a vector from an const array.*
- class `uvector_const_subvector_tlate`  
*Create a const vector from a subvector of another vector.*
- class `ufvector`  
*A vector with unit-stride where the memory allocation is performed in the constructor.*

### Typedefs

- typedef `uvector_tlate< data_t, size_t > uvector`  
*uvector typedef*
- typedef `uvector_view_tlate< data_t, size_t > uvector_view`  
*uvector\_view typedef*
- typedef `uvector_array_tlate< data_t, size_t > uvector_array`  
*uvector\_array typedef*
- typedef `uvector_subvector_tlate< data_t, size_t > uvector_subvector`  
*uvector\_subvector typedef*
- typedef `uvector_const_array_tlate< data_t, size_t > uvector_const_array`  
*uvector\_const\_array typedef*
- typedef `uvector_const_subvector_tlate< data_t, size_t > uvector_const_subvector`  
*uvector\_const\_subvector typedef*
- typedef `uvector_tlate< int > uvector_int`  
*uvector\_int typedef*
- typedef `uvector_view_tlate< int > uvector_int_view`  
*uvector\_int\_view typedef*

- typedef `uvector_array_tlate< int > uvector_int_array`  
*uvector\_int\_array typedef*
- typedef `uvector_subvector_tlate< int > uvector_int_subvector`  
*uvector\_int\_subvector typedef*
- typedef `uvector_const_array_tlate< int > uvector_int_const_array`  
*uvector\_int\_const\_array typedef*
- typedef `uvector_const_subvector_tlate< int > uvector_int_const_subvector`  
*uvector\_int\_const\_subvector typedef*

## Functions

- template<class data\_t>  
std::ostream & `operator<<` (std::ostream &os, const `uvector_view_tlate< data_t > &v`)  
*A operator for naive vector output.*

### 4.17.2 Function Documentation

#### 4.17.2.1 `std::ostream& operator<<` (std::ostream &os, const `uvector_view_tlate< data_t > &v`) [inline]

A operator for naive vector output.

This outputs all of the vector elements. All of these are separated by one space character, though no trailing space or `endl` is sent to the output.

Definition at line 778 of file `uvector_tlate.h`.

## 4.18 `vec_arith.h` File Reference

### 4.18.1 Detailed Description

Vector and matrix arithmetic.

#### Todo

Properly document the operators defined as macros

#### Idea for future

Define operators for complex vector \* real matrix

#### Idea for future

These should be replaced by the BLAS routines where possible

Definition in file `vec_arith.h`.

```
#include <iostream>
#include <complex>
#include <o2scl/cx_arith.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/omatrix_tlate.h>
#include <o2scl/uvector_tlate.h>
#include <o2scl/umatrix_tlate.h>
#include <o2scl/ovector_cx_tlate.h>
```

```
#include <o2scl/omatrix_cx_tlate.h>
#include <o2scl/uvector_cx_tlate.h>
#include <o2scl/umatrix_cx_tlate.h>
```

## Namespaces

- namespace [o2scl\\_arith](#)

## Defines

- #define [O2SCL\\_OP\\_VEC\\_VEC\\_ADD](#)(vec1, vec2, vec3)  
*The header macro for vector-vector addition.*
- #define [O2SCL\\_OP\\_VEC\\_VEC\\_SUB](#)(vec1, vec2, vec3)
- #define [O2SCL\\_OP\\_MAT\\_VEC\\_MULT](#)(vec1, vec2, mat)
- #define [O2SCL\\_OP\\_CMAT\\_CVEC\\_MULT](#)(vec1, vec2, mat)
- #define [O2SCL\\_OP\\_VEC\\_MAT\\_MULT](#)(vec1, vec2, mat)
- #define [O2SCL\\_OP\\_TRANS\\_MULT](#)(vec1, vec2, mat)
- #define [O2SCL\\_OP\\_DOT\\_PROD](#)(dtype, vec1, vec2)
- #define [O2SCL\\_OP\\_CX\\_DOT\\_PROD](#)(dtype, vec1, vec2)
- #define [O2SCL\\_OP\\_SCA\\_VEC\\_MULT](#)(dtype, vecv, vec)
- #define [O2SCL\\_OP\\_VEC\\_SCA\\_MULT](#)(dtype, vecv, vec)
- #define [O2SCL\\_OP\\_VEC\\_VEC\\_PRO](#)(vec1, vec2, vec3)
- #define [O2SCL\\_OPSRC\\_VEC\\_VEC\\_ADD](#)(vec1, vec2, vec3)
- #define [O2SCL\\_OPSRC\\_VEC\\_VEC\\_SUB](#)(vec1, vec2, vec3)
- #define [O2SCL\\_OPSRC\\_MAT\\_VEC\\_MULT](#)(vec1, vec2, mat)
- #define [O2SCL\\_OPSRC\\_CMAT\\_CVEC\\_MULT](#)(vec1, vec2, mat)
- #define [O2SCL\\_OPSRC\\_VEC\\_MAT\\_MULT](#)(vec1, vec2, mat)
- #define [O2SCL\\_OPSRC\\_TRANS\\_MULT](#)(vec1, vec2, mat)
- #define [O2SCL\\_OPSRC\\_DOT\\_PROD](#)(dtype, vec1, vec2)
- #define [O2SCL\\_OPSRC\\_CX\\_DOT\\_PROD](#)(dtype, vec1, vec2)
- #define [O2SCL\\_OPSRC\\_SCA\\_VEC\\_MULT](#)(dtype, vecv, vec)
- #define [O2SCL\\_OPSRC\\_VEC\\_SCA\\_MULT](#)(dtype, vecv, vec)
- #define [O2SCL\\_OPSRC\\_VEC\\_VEC\\_PRO](#)(vec1, vec2, vec3)

## Functions

- template<class vec\_t, class vec2\_t>  
void [vector\\_copy](#) (size\_t N, vec\_t &v, vec2\_t &v2)  
*Naive vector copy.*
- template<class mat\_t, class mat2\_t>  
void [matrix\\_copy](#) (size\_t M, size\_t N, mat\_t &m, mat2\_t &m2)  
*Naive matrix copy.*
- template<class vec\_t, class vec2\_t>  
void [vector\\_cx\\_copy](#) (size\_t N, vec\_t &v, vec2\_t &v2)  
*Naive complex vector copy.*
- template<class mat\_t, class mat2\_t>  
void [matrix\\_cx\\_copy](#) (size\_t M, size\_t N, mat\_t &m, mat2\_t &m2)  
*Naive complex matrix copy.*

## 4.18.2 Define Documentation

### 4.18.2.1 #define O2SCL\_OP\_CMAT\_CVEC\_MULT(vec1, vec2, mat)

#### Value:

```
vec1 operator* \
(const mat &m, const vec2 &x);
```

#### 4.18.2.2 #define O2SCL\_OP\_CX\_DOT\_PROD(dtype, vec1, vec2)

**Value:**

```
dtype dot      \  
(const vec1 &x, const vec2 &y);
```

#### 4.18.2.3 #define O2SCL\_OP\_DOT\_PROD(dtype, vec1, vec2)

**Value:**

```
dtype dot      \  
(const vec1 &x, const vec2 &y);
```

#### 4.18.2.4 #define O2SCL\_OP\_MAT\_VEC\_MULT(vec1, vec2, mat)

**Value:**

```
vec1 operator* \  
(const mat &m, const vec2 &x);
```

#### 4.18.2.5 #define O2SCL\_OP\_SCA\_VEC\_MULT(dtype, vecv, vec)

**Value:**

```
vec operator* \  
(const dtype &x, const vecv &y);
```

#### 4.18.2.6 #define O2SCL\_OP\_TRANS\_MULT(vec1, vec2, mat)

**Value:**

```
vec1 trans_mult \  
(const vec2 &x, const mat &m);
```

#### 4.18.2.7 #define O2SCL\_OP\_VEC\_MAT\_MULT(vec1, vec2, mat)

**Value:**

```
vec1 operator* \  
(const vec2 &x, const mat &m);
```

#### 4.18.2.8 #define O2SCL\_OP\_VEC\_SCA\_MULT(dtype, vecv, vec)

**Value:**

```
vec operator* \  
(const vecv &x, const dtype &y);
```

---



**4.18.2.9 #define O2SCL\_OP\_VEC\_VEC\_ADD(vec1, vec2, vec3)****Value:**

```
vec1 operator+ \
    (const vec2 &x, const vec3 &y);
```

The header macro for vector-vector addition.

Definition at line 91 of file vec\_arith.h.

**4.18.2.10 #define O2SCL\_OP\_VEC\_VEC\_PRO(vec1, vec2, vec3)****Value:**

```
vec1 pair_prod \
    (const vec2 &x, const vec3 &y);
```

**4.18.2.11 #define O2SCL\_OP\_VEC\_VEC\_SUB(vec1, vec2, vec3)****Value:**

```
vec1 operator- \
    (const vec2 &x, const vec3 &y);
```

## 5 O2scl Page Documentation

### 5.1 Pre-Subversion Change Log

**2007-04-29** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Uploaded to subversion (Not publicly accessible)

**2007-04-28** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Created [multi\\_min\\_fix](#) and [fit\\_fix](#) which allow fixing of some parameters before fitting or minimizing
- Created class [pinside](#), to find the point inside a polygon
- Updated some of the documentation
- New command "select-rows" for command-line utility acol, and updated help text.
- Added sum, average, variance, std. dev. functions to [src/base/array.h](#)

**2007-04-15** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Updated [text\\_in\\_file](#) and [text\\_out\\_file](#)
- Updated part, fermion and other related classes to more correctly handle antiparticles, interactions and the possible inclusion of the rest mass. Related testing and documentation was also updated.

**2007-03-25** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Documentation fixes throughout

- Minor changes to several classes
- Reworked the global vector and matrix operators (see [vec\\_arith.h](#))
- Added some complex arithmetic support for `gsl_complex` (see [cx\\_arith.h](#))
- Fixed a few bugs in [ovector\\_cx\\_tlate.h](#)

**2007-03-19** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- A few quick fixes corresponding to the changes between GSL 1.8 and 1.9
- Added and tested new minimizer [gsl\\_mmin\\_bfgs2](#)

**2007-03-08** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Fixed a couple missing distribution files in `src/osp` and `src/oiml`
- Renamed functions `getc()` for compatibility with older systems
- Reworked the testing for the polynomial solving classes

**2007-03-06** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Rewrote [gsl\\_deriv](#) to handle function objects more cleanly.
- Tested with GSL version 1.9.

**2007-03-03** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Slight rewrites of sparse matrix methods
- Added a couple of array functions and repaired a few header files
- Added some tests in `src/base`
- Fixed `Matrix*vector` functions which were incorrect (need more tests on these)
- Began improvements for how array-indexing errors are handled

**2007-02-24** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- New version 0.77
- Modifications, new testing, and better documentation for sparse matrix implementation
- Added ODE solver using iterative matrix methods, [ode\\_it\\_solve](#)
- Now properly using doxygen-1.5.1
- Fixed a bug which appeared in `src/base/fpwrap.h`

**2007-02-20** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Fixed missing `~funct_nopar_fptr()`
  - Added `umatrix` objects
  - Made a couple changes in `schematic_eos`
-

**2007-02-15** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Fixed `iohb_base.cpp` which caused typecasting warnings. There are hopefully now no warnings even when compiled with `gcc -Wall -ansi -pedantic`

**2007-02-13** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Improved `file_detect` so that file reading does not fail if the filename contains extra whitespace.

**2007-02-10** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Major updates for the `tov_solve` class.
- Fixed problem in which 'make' failed.

**2007-02-06** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Corrected `configure.ac` a bit.
- Made `adapt_step::astep()` virtual.

**2007-02-04** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Tested on Cygwin
- Renamed some of the `fpwrap` function to camel case for consistency with `FunctionParser`.
- Made the `fpwrap` object in the `table` class public (for now). There are still issues with const-correctness for the `fpwrap` and `table` objects.

**2007-02-01** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Made a 'tov\_solve2' class so I could work on updating the TOV solver

**2007-01-30** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Added Lanczos diagonalization in 'other' section
- Corrected some `int`'s which should be `size_t`'s

**2007-01-25** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Added a couple commands to 'acol'.
- Fixed `src/oiml/t*.cpp` not appearing in the distribution
- Tested on Mac OS X
- Updated documentation a little

**2007-01-23** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- Added a function to compute the "slope parameter for the symmetry energy" to `hadronic_eos`.

**2007-01-12** Andrew W. Steiner <[steinera@pa.msu.edu](mailto:steinera@pa.msu.edu)>

- All 113 tests passed.
-

## 5.2 Todo List

page [O2scl User's Guide](#) Finish this list

page [O2scl User's Guide](#) Redo this section since we've added [string\\_conv.h](#)

Class [bin\\_size](#) Doesn't seem to be working yet.

Class [cern\\_adapt](#) It looks like the first segment is of zero size, is this because there's an offset? Double check that we don't have memory issues if nseg=nsub.

Class [collection](#)

- If pointer\_in gets a null pointer it does nothing. Should we replace this behaviour by two pointer\_in() functions. One which does nothing if it gets a null pointer, and one which will go ahead and set the pointer to null. This is useful for output object which have default values to be used if they are given a null pointer.
- More testing on rewrite() function.
- Think more about adding arrays of pointers? pointers to arrays?
- Modify static data output so that if no objects of a type are included, then no static data is output for that type? (No, it's too hard to go through all objects looking for an object of a particular type).

Class [columnify](#) Move the [screenify\(\)](#) functionality from [misc.h](#) into this class?

Class [contour](#)

- Some contours which should be closed are not properly closed yet. See the tests for examples which fail.
- Use [twod\\_intp](#) for regrid\_data
- Work on how memory is allocated
- Include the functionality to provide regions to be shaded instead of just lines. This can be done by providing a method, i.e. regions() which converts the curves into regions.

Class [eqi\\_deriv](#) The uncertainties in the derivatives are not yet computed and the second and third derivative formulas are not yet finished.

Global [eqi\\_deriv::deriv\\_array\(size\\_t nv, double dx, const vec\\_t &y, vec\\_t &dydx\)](#) generalize to other values of npoints.

Class [gaussian\\_2d](#) Double check that sigma is implemented correctly

Class [gsl\\_anneal](#) Implement different stepsizes for the different dimensions

Class [gsl\\_astep](#) Finish implementing the scaled "control"

Class [gsl\\_astep](#) Allow user to find out how many steps were taken, etc.

---

Global `gsl_cubic_real_coeff::gsl_poly_complex_solve_cubic2(double a, double b, double c, gsl_complex *z0, gsl_complex *z1, gsl_complex *z2)` Demonstrate that this works better than the original GSL version.

Class `gsl_fit` Properly generalize other vector types than `ovector_view`

Class `gsl_fit` Allow the user to specify the derivatives

Class `gsl_fit` Fix so that the user can specify automatic scaling of the fitting parameters, where the initial guess are used for scaling so that the fitting parameters are near unity.

Class `gsl_inte` Move the documentation below to a more sensible place

Class `gsl_inte_qag` Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Class `gsl_inte_qagi` I had to add extra code to check for non-finite values for some integrations. This should be checked.

Class `gsl_inte_qawf_cos` Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Class `gsl_inte_qawo_cos` Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Class `gsl_inte_singular::extrapolation_table` Move this to a new class, with `qelg()` as a method

Class `gsl_inte_table` Move `gsl_integration_workspace` to a separate class and remove this class, making all children direct descendants of `gsl_inte` instead. We'll have to figure out what to do with the data member `workspace` though.

Class `gsl_min_brent` Simplify temporary storage and document stopping conditions.

Class `gsl_mmin_conf` A bit of needless copying is required in the function wrapper to convert from `gsl_vector` to the templated vector type. This can be fixed, probably by rewriting `take_step` to produce a `vec_t &x1` rather than a `gsl_vector *x1`;

Global `gsl_mmin_conf::it_info` Document this better

Class `gsl_mmin_conp` A bit of needless copying is required in the function wrapper to convert from `gsl_vector` to the templated vector type. This can be fixed.

Class `gsl_mmin_conp` Document stopping conditions

Class `gsl_mmin_simp` Not properly generalized to non-GSL vectors (to do this, I'll have to store the simplex as a `vec_t` array rather than a `gsl_matrix`). Right now, this only works with `vec_t = ovector_view`.

---

**Class `gsl_mmin_simp`** Add a `minimize` function which allows specification of the entire simplex.

**Class `gsl_mmin_simp`** Gracefully ensure memory allocation and deallocation is performed automatically.

**Class `gsl_mmin_simp`** Test `mmin_twovec()`.

**Class `gsl_mmin_simp`** Document how `set()` chooses the simplex from the initial guess and step size

**Class `gsl_mroot_hybrids`** Internally, there is a little unnecessary copying back and forth of vectors

**Class `gsl_quartic_real2`** Document the distinction between this class and `gsl_quartic_real`

**Class `gsl_root_brent`** There is some duplication in the variables `x_lower`, `x_upper`, `a`, and `b`, which could be removed.

**Class `gsl_series`** Covert to use a more general vector

**Class `hybrids_base`** Document the individual functions for this class

**Class `io_base`** Should the `remove()` functions be moved to class `collection`?

**Global `minimize::bracket(double &ax, double &bx, double &cx, double &fa, double &fb, double &fc, param_t &pa, func_t &func)`**  
Improve this algorithm with the standard golden ratio method.

**Global `minimize::bracket(double &ax, double &bx, double &cx, double &fa, double &fb, double &fc, param_t &pa, func_t &func)`**  
Double check that this works when at least two of `f(a)`, `f(b)` and `f((a+b)/2)` are equal.

**Class `naive_metropolis`** Talk about how accurate this is with  $\beta$ .

**Class `naive_metropolis`** Redo statistics, talk about how many times the integral is evaluated.

**Class `naive_metropolis`** Offer an option of giving more results than just the final average and error?

**Class `naive_quartic_real`** 3/8/07 - Compilation at the NSCL produced non-finite values in `solve_r()` for some values of the coefficients. This should be checked.

**Class `naive_quartic_real`** It looks like this code is tested only for `a4=1`, and if so, the tests should be generalized.

**Class `naive_quartic_real`** Also, there is a hard-coded number in here ( $10^{-6}$ ), which might be moved to a data member?

**Class `o2scl_interp_vec`** Need to fix constructor to behave properly if `init()` fails. It should free the memory and set `ln` to zero.

**Class `ode_it_solve`** Max and average tolerance?

**Class `ode_it_solve`** partial correction option?

**Class `ode_iv_solve`** Consider modifying so that this can handle tables which are too small by removing `o2scl` the rows and doubling the stepsize.

**Class `ode_iv_solve`** Decide what to do if the adaptive stepper fails. (1/18/07

- two approaches: continue no matter what, or just stop)

**Class `ode_iv_solve`** Convert to using `astep_derivs()`?

**Class `omatrix_view_tlate`** This class isn't sufficiently general for some applications, such as sub-matrices of higher-dimensional structures. It might be nice to create a more general class with a "stride" and a "tda".

**Class `other_todos_and_bugs`** • Fix problems with `-ansi` compilation on Cygwin

- More examples and benchmarks
- There is a problem with const-correctness in vectors. I'm not sure how it's best solved. It could be best to create two kinds of `ovector_view`'s: one const and one not.
- Document a list of all global functions and operators
- Make sure we have `uvector_alloc`, `uvector_cx_alloc`, `ovector_cx_const_reverse`, `ovector_cx_const_subvector_reverse`, `uvector_reverse`, `uvector_const_reverse`, `uvector_subvector_reverse`, `uvector_const_subvector_reverse`, `omatrix_cx_diag`, `blah_const_diag`, `umatrix_diag`, and `umatrix_cx_diag`
- `ovector_cx_view::operator=(uvector_cx_view &)` is missing
- `ovector_cx::operator=(uvector_cx_view &)` is missing
- `uvector_c_view::operator+=(complex)` is missing
- `uvector_c_view::operator-=(complex)` is missing
- `uvector_c_view::operator*=(complex)` is missing

**Class `ovector_cx_tlate`** There is a slight difference between how this works in comparison to `MV++`. The function `allocate()` operates a little differently than `newsize()`, as it will feel free to allocate new memory when `owner` is false. It's not clear if this is an issue, however, since it doesn't appear possible to create an `ovector_cx_tlate` with a value of `owner` equal to zero. This situation ought to be clarified further.

**Class `ovector_cx_tlate`** Add `subvector_stride`, `const_subvector_stride`

**Class `ovector_cx_view_tlate`** Move conversion b/w `complex<double>` and `gsl_complex` to `cx_arith.h`

**Class `polylog`** • Give error estimate?

---

- Improve accuracy?
- Use more sophisticated interpolation?
- Add the series  $Li(n, x) = x + 2^{-n}x^2 + 3^{-n}x^3 + \dots$  for  $x \rightarrow 0$ ?
- Implement for positive arguments  $< 1.0$
- Make another `polylog` class which implements series acceleration?

Global `rnga::clock_seed()` Ensure this function is ANSI compatible

Class `search_vec` The documentation here is still kind of unclear.

Class `simple_jacobian` Double check that this class works with arrays

Global `smart_interp::find_subset(const double a, const double b, size_t sz, const vec_t &x, const vec_t &y, size_t &nsz, bool &increasing)`  
After row and row2 are set, check to make sure the entire inside region is monotonic before expanding

Class `table` Move the discussion above to the user guide?

Class `table` Add `interp()` and related functions which avoid caching and can thus be const (This has been started with `interp_const()`)

Global `table::set_nlines_auto(size_t il)` Resolve whether `set()` should really use this approach. Also, resolve whether this should replace `set_nlines()` (It could be that the answer is no, because as the documentation in the other version states, the other version is useful if you have columns not owned by the `table`.)

Global `table::new_column(std::string name, ovector_view *ldat)` We've got to figure out what to do if `ldat` is too small. If it's smaller than `nlines`, obviously we should just fail, but what if it's size is between `nlines` and `maxlines`?

Global `text_out_file::text_out_file(std::ostream *out_file, int width=80)` Ensure streams are not opened in binary mode for safety.

Class `timer_gettod` Better testing which doesn't use a fixed number of mathematical operations, but automatically selects enough operations.

Class `twod_intp` Test non-square data

Class `twod_intp` Allow specification of data as `[ny][nx]` or as `[nx][ny]`?

Class `twod_intp` Need to rewrite to use `o2scl_interp` after that class is finished.

Class `twod_intp` Could also include mixed second/first derivatives: `deriv_xxy` and `deriv_xyy`.

---



**Class `uvector_cx_view_tlate`** Write `lookup()` method, and possible an `erase()` method.

**File `array.h`** Ensure that `array_row` works, either here or in `src/ode/ode_it_solve_ts.cpp`

**Global `array_2d_out`** If all of the matrix elements are positive integers and scientific mode is not set, then we can avoid printing the extra spaces.

**Global `matrix_out`** Might need to test to see what happens if all of the matrix elements are positive integers and scientific mode is not set.

**File `cx_arith.h`** Define operators with assignment for complex + double

**File `cx_arith.h`** Ensure all the trig functions are tested

**Global `err_assert`** Should make this consistent with `assert()` using `NDEBUG`

**Global `screenify`** Consider making this into a class so that the memory for the output columns is automatically handled.

**Global `operator<<`** Maybe remove this function, as it's superceded by `matrix_out()`?

**Global `operator<<`** This assumes that scientific mode is on and `showpos` is off. It'd be nice to fix this.

**Global `double_to_latex`** Fix to ensure final zeros are printed properly if requested

**Global `dtos`** Add error checking to this function using `if(strout << x)`

**Global `operator<<`** This assumes that scientific mode is on and `showpos` is off. It'd be nice to fix this.

**Global `operator<<`** This assumes that scientific mode is on and `showpos` is off. It'd be nice to fix this.

**File `vec_arith.h`** Properly document the operators defined as macros

## 5.3 Download O2scl

The present version is 0.8, which can be obtained from

- [http://sourceforge.net/project/showfiles.php?group\\_id=206918](http://sourceforge.net/project/showfiles.php?group_id=206918)

---

The previous versions are at

- <http://tharkad.pa.msu.edu/~asteiner/o2scl/hal-0.78.tar.gz>
  - <http://tharkad.pa.msu.edu/~asteiner/o2scl/hal-0.77.tar.gz>
  - <http://tharkad.pa.msu.edu/~asteiner/o2scl/hal-0.76.tar.gz>
-

## 5.4 Ideas for future development

**Class `base_interp`** These might work for decreasing functions by just replacing calls to `search_vec::bsearch_inc()` with `search_vec::bsearch_dec()`. If this is the case, then this should be rewritten accordingly.

**Class `cern_adapt`** More error checking, e.g. ensure the user doesn't try to get a segment greater than the total number of segments.

**Class `cern_adapt`** Allow user to set the initial segments?

**Class `cern_mroot`** Modify this so it handles functions which return non-zero values.

**Class `cern_mroot_root`** Double-check this class to make sure it cannot fail while returning 0 for success.

**Class `deriv`** Improve the methods for second and third derivatives

**Class `file_detect`** Allow the user to specify the compression commands in configure, or at least specify the path to `gzip`, `bzip2`, etc.

**Class `gsl_anneal`** Implement a more general routine which would allow the solution of discrete problems like the Traveling Salesman problem.

**Class `gsl_deriv`** Include the forward and backward GSL derivatives

**Class `gsl_mmin_wrapper`** There's a bit of extra vector copying here which could potentially be avoided.

**Class `lanczos`** The function `eigen_tdiag()` automatically sorts the eigenvalues, which may not be necessary.

**Class `planar_intp`** Rewrite so that it never fails unless all the points in the data set lie on a line. This would probably demand sorting all of the points by distance from desired location.

**Class `table`** The `nlines` vs. `maxlines` and automatic resizing of table-owned vs. user-owned vectors could be reconsidered, especially now that `ovectors` can automatically resize on their own.

**Class `test_mgr`** `test_mgr::success` and `test_mgr::last_fail` should be protected, but that breaks the `operator+()` function. Can this be fixed?

**File `vec_arith.h`** Define operators for complex vector \* real matrix

**File `vec_arith.h`** These should be replaced by the BLAS routines where possible

---

## 5.5 Bug List

**Class `collection`** • Ensure that the user cannot add a object with a name of ptrXXX.

- Test\_type does not test handle static data or pointers.
- Check strings and words for characters that we can't handle
- The present version of a text-file requires strings to contain at least one printable character.
- Ensure that all matching is done by both type and name if possible.

**Class `other_todos_and_bugs`** • The file configure.ac does not correctly produce an error if the argument `--disable-readline` is not given when the libncurses and libreadline libraries are not present.

- BLAS libraries not named `libblas` or `libgslblas` are not properly detected in `./configure` and will have to be added manually.
- The `-lm` flag may not be added properly by `./configure`

**Class `quad_intp`** This class doesn't seem to work at present.

---

# Index

- adapt\_step, 66
  - astep, 67
  - astep\_derivs, 67
  - set\_step, 68
- add\_col\_from\_table
  - table, 385
- add\_type
  - io\_manager, 241
- align
  - columnify, 108
- array.h, 423
  - vector\_avg, 424
  - vector\_out, 425
  - vector\_sum, 425
  - vector\_variance, 425
- array\_2d\_alloc, 68
- array\_2d\_out
  - columnify.h, 426
- array\_abort
  - err\_class, 137
- array\_alloc, 68
- array\_const\_reverse, 69
- array\_const\_subvector, 69
- array\_const\_subvector\_reverse, 70
- array\_interp, 71
- array\_interp\_vec, 71
- array\_reverse, 72
- array\_row, 72
- array\_subvector, 73
- array\_subvector\_reverse, 73
- astep
  - adapt\_step, 67
  - gsl\_astep, 165
  - nonadapt\_step, 292
- astep\_derivs
  - adapt\_step, 67
  - gsl\_astep, 165
  - nonadapt\_step, 292
- base\_interp, 74
  - min\_size, 75
- base\_ioc, 75
- bin\_size, 76
  - calc\_bin, 77
- binary\_in\_file, 77
- binary\_out\_file, 78
- binary\_to\_hex
  - misc.h, 434
- bool\_io\_type, 80
- bracket
  - minimize, 260
- bsearch\_dec
  - search\_vec, 368
- bsearch\_inc
  - search\_vec, 368
- calc
  - deriv, 130
- calc2\_array
  - eqi\_deriv, 135
- calc3\_array
  - eqi\_deriv, 135
- calc\_array
  - eqi\_deriv, 135
- calc\_bin
  - bin\_size, 77
- calc\_contours
  - contour, 121
- calc\_err\_int
  - deriv, 130
- calc\_int
  - deriv, 130
- capacity
  - ovector\_view\_tlate, 348
- central\_deriv
  - gsl\_deriv, 170
- cern\_adapt, 80
  - nseg, 82
- cern\_cauchy, 82
- cern\_cubic\_real\_coeff, 83
- cern\_deriv, 84
- cern\_gauss, 85
- cern\_gauss56, 87
- cern\_minimize, 87
  - min\_bkt, 88
  - set\_delta, 89
- cern\_mroot, 89
  - eps, 91
  - get\_info, 90
  - maxf, 91
- cern\_mroot\_root, 91
  - eps, 93
  - get\_info, 93
  - maxf, 93
- cern\_quartic\_real\_coeff, 94
  - solve\_rc, 94
- cern\_root, 94
  - mode, 96
  - set\_mode, 95
- cerr\_print
  - err\_hnd.h, 430
- ch\_owner
  - table, 385
- char\_io\_type, 96
  - getcc, 96
- cinput, 97

- clear\_data
  - table, 387
- clear\_types
  - io\_type\_info, 247
- cli, 98
  - gnu\_intro, 100
  - set\_alias, 100
  - set\_comm\_option, 100
  - set\_parameters, 100
  - set\_verbose, 100
- clock\_seed
  - rnga, 365
- cmd\_line\_arg, 100
- col, 101
- collection, 101
  - disown, 104
  - in\_one, 105
  - in\_one\_name, 105
  - out\_one, 105
  - remove, 105
  - rewrite, 104
- collection::iterator, 105
- collection::type\_iterator, 106
- collection\_entry, 107
- cols
  - omatrix\_cx\_view\_tlate, 318
  - omatrix\_view\_tlate, 322
  - umatrix\_cx\_view\_tlate, 406
  - umatrix\_view\_tlate, 410
- columnify, 108
  - align, 108
- columnify.h, 425
  - array\_2d\_out, 426
  - matrix\_out, 426
- comm\_option, 109
- comm\_option\_funct, 109
- comm\_option\_mfptr, 110
- comm\_option\_s, 111
- comp\_gen\_inte, 112
  - set\_ptrs, 113
- comp\_gen\_inte::od\_parms, 113
- composite\_inte, 114
  - set\_ptrs, 115
- composite\_inte::od\_parms, 115
- constraint
  - minimize.h, 432
- cont\_constraint
  - minimize.h, 432
- cont\_lower\_bound
  - minimize.h, 433
- contour, 116
  - calc\_contours, 121
  - get\_contour, 121
  - get\_data, 121
  - get\_edges\_for\_level, 122
  - is\_point\_inside\_old, 122
  - lines\_cross\_old, 122
  - regrid\_data, 121
  - set\_data, 121
  - set\_levels, 121
  - smooth\_contours, 122
- contract\_by\_best
  - gsl\_mmin\_simp, 210
- count\_words
  - misc.h, 434
- coutput, 122
  - npointers, 123
  - pointer\_lookup, 123
- coutput::ltptr, 124
- cspline\_interp, 124
  - init, 125
- cubic
  - gsl\_mmin\_linmin, 208
- cubic\_complex, 125
  - solve\_c, 126
- cubic\_real, 126
- cubic\_real\_coeff, 127
- cubic\_std\_complex, 128
  - solve\_c, 128
- cx\_arith.h, 426
- delete\_column
  - table, 387
- deriv, 129
  - calc, 130
  - calc\_err\_int, 130
  - calc\_int, 130
  - gsl\_chebapp, 167
  - table, 386
- deriv2
  - table, 386
- deriv::dpars, 131
- deriv\_array
  - eqi\_deriv, 135
- deriv\_eps
  - root, 366
- deriv\_ioc, 131
- disown
  - collection, 104
- double\_io\_type, 132
- double\_to\_html
  - string\_conv.h, 444
- double\_to\_ieee\_string
  - string\_conv.h, 444
- double\_to\_latex
  - string\_conv.h, 444
- dtos
  - string\_conv.h, 445
- e2\_gaussian
  - o2scl\_const, 60

- e2\_hlorentz
    - o2scl\_const, 60
  - e2\_mkssa
    - o2scl\_const, 61
  - eigen\_tdiag
    - lanczos, 252
  - eigenvalues
    - lanczos, 252
  - eps
    - cern\_mroot, 91
    - cern\_mroot\_root, 93
  - eqi\_deriv, 132
    - calc2\_array, 135
    - calc3\_array, 135
    - calc\_array, 135
    - deriv\_array, 135
    - set\_npoints, 134
    - set\_npoints2, 134
  - err\_assert
    - err\_hnd.h, 430
  - err\_class, 135
    - array\_abort, 137
    - gsl\_hnd, 137
  - err\_hnd.h
    - gsl\_continue, 430
    - gsl\_ebadfunc, 430
    - gsl\_ebadlen, 431
    - gsl\_ebadtol, 431
    - gsl\_ecache, 431
    - gsl\_ediverge, 431
    - gsl\_edom, 430
    - gsl\_efactor, 430
    - gsl\_efailed, 430
    - gsl\_efault, 430
    - gsl\_efilenotfound, 431
    - gsl\_einval, 430
    - gsl\_eloss, 431
    - gsl\_emaxiter, 431
    - gsl\_enomem, 430
    - gsl\_enoprog, 431
    - gsl\_enoprogj, 431
    - gsl\_enotsqr, 431
    - gsl\_eof, 431
    - gsl\_eovrflw, 431
    - gsl\_erange, 430
    - gsl\_eround, 431
    - gsl\_erunaway, 430
    - gsl\_esanity, 430
    - gsl\_esing, 431
    - gsl\_etable, 431
    - gsl\_etol, 431
    - gsl\_etolf, 431
    - gsl\_etolg, 431
    - gsl\_etolx, 431
    - gsl\_eundrflw, 431
    - gsl\_eunimpl, 431
    - gsl\_eunsup, 431
    - gsl\_ezerodiv, 431
    - gsl\_failure, 430
    - gsl\_index, 431
    - gsl\_memtype, 431
    - gsl\_nobase, 431
    - gsl\_notfound, 431
    - gsl\_success, 430
  - err\_hnd.h, 428
    - cerr\_print, 430
    - err\_assert, 430
    - err\_print, 430
  - err\_print
    - err\_hnd.h, 430
  - exact\_jacobian, 137
  - exact\_jacobian::ej\_parms, 138
  - fermi\_function
    - misc.h, 434
  - feval
    - gsl\_inte\_qng, 192
  - file\_detect, 139
    - open, 140
  - find\_subset
    - smart\_interp, 374
  - fit
    - fit\_base, 141
    - fit\_fix\_pars, 142
    - gsl\_fit, 173
    - min\_fit, 257
  - fit\_base, 140
    - fit, 141
    - print\_iter, 141
  - fit\_fix\_pars, 141
    - fit, 142
  - fit\_funct, 143
  - fit\_funct\_fptr, 143
  - fit\_funct\_mfp\_ptr, 144
  - fit\_vfunkt, 145
  - fit\_vfunkt\_fptr, 146
  - fit\_vfunkt\_mfp\_ptr, 147
  - free
    - omatrix\_cx\_tlate, 316
    - omatrix\_tlate, 320
    - ovector\_cx\_tlate, 339
    - ovector\_tlate, 345
    - umatrix\_cx\_tlate, 404
    - umatrix\_tlate, 408
    - uvector\_cx\_tlate, 416
    - uvector\_tlate, 420
  - funct, 147
  - funct\_fptr, 148
  - funct\_fptr\_noerr, 149
  - funct\_fptr\_nopar, 150
  - funct\_mfp\_ptr, 150
-

---

funct\_mfp\_ptr\_noerr, 151  
funct\_mfp\_ptr\_nopar, 152

gaussian\_2d, 153  
gen\_inte, 153  
    get\_error, 154  
gen\_test\_number, 154  
get\_coefficient  
    gsl\_chebapp, 168  
get\_column  
    table, 383, 384  
get\_contour  
    contour, 121  
get\_data  
    contour, 121  
get\_edges\_for\_level  
    contour, 122  
get\_error  
    gen\_inte, 154  
    inte, 238  
    multi\_inte, 279  
get\_gsl\_rng  
    gsl\_rnga, 229  
get\_info  
    cern\_mroot, 90  
    cern\_mroot\_root, 93  
getcc  
    char\_io\_type, 96  
gm\_parms, 155, 156  
gnu\_intro  
    cli, 100  
grad\_funct, 157  
grad\_funct\_fptr, 157  
grad\_funct\_mfp\_ptr, 158  
grad\_vfunct, 159  
grad\_vfunct\_fptr, 159  
grad\_vfunct\_mfp\_ptr, 160  
gradient, 161  
gradient\_array, 161  
gsl\_anneal, 162  
gsl\_astep, 163  
    astep, 165  
    astep\_derivs, 165  
gsl\_astep::gsl\_ode\_control, 165  
gsl\_astep::gsl\_odeiv\_evolve, 166  
gsl\_cgs, 42  
gsl\_cgsm, 46  
gsl\_chebapp, 166  
    deriv, 167  
    get\_coefficient, 168  
    init, 167  
    inte, 167  
    set\_order, 167  
gsl\_continue  
    err\_hnd.h, 430  
gsl\_cubic\_real\_coeff, 168  
    gsl\_poly\_complex\_solve\_cubic2, 168  
gsl\_deriv, 169  
    central\_deriv, 170  
    h, 170  
    h\_opt, 170  
gsl\_ebadfunc  
    err\_hnd.h, 430  
gsl\_ebadlen  
    err\_hnd.h, 431  
gsl\_ebadtol  
    err\_hnd.h, 431  
gsl\_ecache  
    err\_hnd.h, 431  
gsl\_ediverge  
    err\_hnd.h, 431  
gsl\_edom  
    err\_hnd.h, 430  
gsl\_efactor  
    err\_hnd.h, 430  
gsl\_efailed  
    err\_hnd.h, 430  
gsl\_efault  
    err\_hnd.h, 430  
gsl\_efilenotfound  
    err\_hnd.h, 431  
gsl\_einval  
    err\_hnd.h, 430  
gsl\_eloss  
    err\_hnd.h, 431  
gsl\_emaxiter  
    err\_hnd.h, 431  
gsl\_enomem  
    err\_hnd.h, 430  
gsl\_enoprog  
    err\_hnd.h, 431  
gsl\_enoproj  
    err\_hnd.h, 431  
gsl\_enotsqr  
    err\_hnd.h, 431  
gsl\_eof  
    err\_hnd.h, 431  
gsl\_eovrflw  
    err\_hnd.h, 431  
gsl\_erange  
    err\_hnd.h, 430  
gsl\_eround  
    err\_hnd.h, 431  
gsl\_erunaway  
    err\_hnd.h, 430  
gsl\_esanity  
    err\_hnd.h, 430  
gsl\_esing  
    err\_hnd.h, 431  
gsl\_etable  
    err\_hnd.h, 431

---

- gsl\_etol
  - err\_hnd.h, 431
- gsl\_etolf
  - err\_hnd.h, 431
- gsl\_etolg
  - err\_hnd.h, 431
- gsl\_etolx
  - err\_hnd.h, 431
- gsl\_eundrflw
  - err\_hnd.h, 431
- gsl\_eunimpl
  - err\_hnd.h, 431
- gsl\_eunsup
  - err\_hnd.h, 431
- gsl\_ezerodiv
  - err\_hnd.h, 431
- gsl\_failure
  - err\_hnd.h, 430
- gsl\_fft, 170
- gsl\_fit, 171
  - fit, 173
- gsl\_fit::func\_par, 173
- gsl\_HH\_solver, 174
- gsl\_hnd
  - err\_class, 137
- gsl\_index
  - err\_hnd.h, 431
- gsl\_inte, 174
- gsl\_inte\_cheb, 176
- gsl\_inte\_kronrod, 176
- gsl\_inte\_qag, 177
  - set\_key, 178
- gsl\_inte\_qagi, 179
  - integ, 180
  - integ\_err, 180
- gsl\_inte\_qagil, 180
  - integ, 181
  - integ\_err, 181
- gsl\_inte\_qagiu, 181
  - integ, 183
  - integ\_err, 183
- gsl\_inte\_qags, 183
- gsl\_inte\_qawc, 184
- gsl\_inte\_qawf\_cos, 185
- gsl\_inte\_qawf\_sin, 186
- gsl\_inte\_qawo\_cos, 187
- gsl\_inte\_qawo\_sin, 188
- gsl\_inte\_qaws, 190
- gsl\_inte\_qaws::fn\_qaws\_params, 191
- gsl\_inte\_qng, 192
  - feval, 192
- gsl\_inte\_singular, 193
- gsl\_inte\_singular::extrapolation\_table, 194
- gsl\_inte\_table, 194
  - retrieve, 196
- gsl\_inte\_transform, 196
  - gsl\_integration\_qk\_o2scl, 197
- gsl\_integration\_qk\_o2scl
  - gsl\_inte\_transform, 197
- gsl\_LU\_solver, 197
- gsl\_memtype
  - err\_hnd.h, 431
- gsl\_min\_brent, 197
  - min\_bkt, 199
- gsl\_miser, 199
- gsl\_mks, 50
- gsl\_mkssa, 53
- gsl\_mmin\_base, 200
  - intermediate\_point, 201
  - minimize, 201
- gsl\_mmin\_bfgs2, 202
- gsl\_mmin\_conf, 203
  - it\_info, 206
  - set, 205
  - set\_de, 205
- gsl\_mmin\_conf\_array, 206
- gsl\_mmin\_conp, 206
- gsl\_mmin\_linmin, 207
  - cubic, 208
  - interp\_quad, 208
  - minimize, 208
- gsl\_mmin\_simp, 208
  - contract\_by\_best, 210
  - move\_corner, 210
  - print\_iter, 210
  - print\_simplex, 211
- gsl\_mmin\_simp\_b, 211
  - nmsimplex\_size, 211
- gsl\_mmin\_simp\_b::simp\_state\_t, 212
- gsl\_mmin\_wrap\_base, 212
- gsl\_mmin\_wrapper, 213
- gsl\_monte, 214
- gsl\_mroot\_hybrids, 215
  - iterate, 217
  - set\_de, 217
  - shrink\_step, 217
- gsl\_mroot\_hybrids::o2scl\_hybrid\_state\_t, 218
- gsl\_nobase
  - err\_hnd.h, 431
- gsl\_notfound
  - err\_hnd.h, 431
- gsl\_num, 57
- gsl\_poly\_complex\_solve\_cubic2
  - gsl\_cubic\_real\_coeff, 168
- gsl\_poly\_real\_coeff, 218
  - solve\_rc, 219
- gsl\_QR\_solver, 220
- gsl\_quadratic\_real\_coeff, 220
- gsl\_quartic\_real, 221
  - solve\_r, 221



- gsl\_quartic\_real2, 222
    - solve\_r, 222
  - gsl\_rk8pd, 222
    - step, 224
  - gsl\_rk8pd\_fast, 224
    - step, 225
  - gsl\_rkck, 226
    - step, 227
  - gsl\_rkck\_fast, 227
    - step, 228
  - gsl\_rnga, 228
    - get\_gsl\_rng, 229
  - gsl\_root\_brent, 229
    - iterate, 231
    - set, 231
    - solve\_bkt, 231
  - gsl\_root\_stef, 231
    - iterate, 232
    - set, 232
  - gsl\_series, 233
  - gsl\_success
    - err\_hnd.h, 430
  - gsl\_vegas, 233
  - h
    - gsl\_deriv, 170
  - h\_opt
    - gsl\_deriv, 170
  - hybrids\_base, 234
  - in\_file\_format, 235
  - in\_one
    - collection, 105
  - in\_one\_name
    - collection, 105
  - init
    - cspline\_interp, 125
    - gsl\_chebapp, 167
  - init\_column
    - table, 385
  - inside
    - pinside, 350
  - int\_io\_type, 236
  - inte, 237
    - get\_error, 238
    - gsl\_chebapp, 167
    - integ\_err, 238
  - integ
    - gsl\_inte\_qagi, 180
    - gsl\_inte\_qagil, 181
    - gsl\_inte\_qagiu, 183
    - table, 386
  - integ\_err
    - gsl\_inte\_qagi, 180
    - gsl\_inte\_qagil, 181
    - gsl\_inte\_qagiu, 183
  - inte, 238
  - intermediate\_point
    - gsl\_mmin\_base, 201
  - interp
    - planar\_intp, 352
    - quad\_intp, 358
    - table, 385, 386
  - interp\_const
    - table, 386
  - interp\_quad
    - gsl\_mmin\_linmin, 208
  - intersect
    - pinside, 349
  - io\_base, 238
    - io\_base, 240
    - pointer\_out, 240
  - io\_manager, 241
    - add\_type, 241
  - io\_tlate, 242
    - object\_in\_mem, 245
    - stat\_input, 245
  - io\_type\_info, 246
    - clear\_types, 247
    - remove\_type, 247
  - io\_vtlate, 247
    - stat\_input, 248
  - is\_owner
    - ovector\_view\_tlate, 348
  - is\_point\_inside\_old
    - contour, 122
  - it\_info
    - gsl\_mmin\_conf, 206
  - iterate
    - gsl\_mroot\_hybrids, 217
    - gsl\_root\_brent, 231
    - gsl\_root\_stef, 232
  - jac\_funct, 248
  - jac\_funct\_fptr, 249
  - jac\_funct\_mfp\_ptr, 249
  - jacobian, 250
  - lanczos, 251
    - eigen\_tdiag, 252
    - eigenvalues, 252
    - product, 252
  - lib\_settings.h, 431
  - lib\_settings\_class, 252
  - linear\_interp, 253
  - linear\_solver, 254
  - lines\_cross\_old
    - contour, 122
  - long\_io\_type, 254
  - lookup
    - ovector\_view\_tlate, 348
    - uvector\_view\_tlate, 421
-

- lookup\_column
  - table, 385
- lookup\_form
  - table, 385
- lower\_bound
  - minimize.h, 433
- mass\_alpha
  - o2scl\_fm, 62
- matrix\_out
  - columnify.h, 426
- maxf
  - cern\_mroot, 91
  - cern\_mroot\_root, 93
- mcarlo\_inte, 255
- min
  - minimize, 259
- min\_bkt
  - cern\_minimize, 88
  - gsl\_min\_brent, 199
  - minimize, 259
- min\_bkt\_de
  - minimize, 259
- min\_de
  - minimize, 259
- min\_fit, 256
  - fit, 257
- min\_fit::func\_par, 257
- min\_size
  - base\_interp, 75
- minimize, 258
  - bracket, 260
  - gsl\_mmin\_base, 201
  - gsl\_mmin\_linmin, 208
  - min, 259
  - min\_bkt, 259
  - min\_bkt\_de, 259
  - min\_de, 259
  - print\_iter, 259
- minimize.h, 432
  - constraint, 432
  - cont\_constraint, 432
  - cont\_lower\_bound, 433
  - lower\_bound, 433
- minteg\_array
  - naive\_metropolis, 289
- minteg\_err
  - naive\_metropolis, 289
- misc.h, 433
  - binary\_to\_hex, 434
  - count\_words, 434
  - fermi\_function, 434
  - screenify, 434
- mm\_funcnt, 260
- mm\_funcnt\_fptr, 261
- mm\_funcnt\_fptr\_nopar, 261
- mm\_funcnt\_gsl, 262
- mm\_funcnt\_mfptr, 263
- mm\_funcnt\_mfptr\_nopar, 264
- mm\_vfuncnt, 265
- mm\_vfuncnt\_fptr, 266
- mm\_vfuncnt\_fptr\_nopar, 266
- mm\_vfuncnt\_gsl, 267
- mm\_vfuncnt\_mfptr, 268
- mm\_vfuncnt\_mfptr\_nopar, 269
- mode
  - cern\_root, 96
- move\_corner
  - gsl\_mmin\_simp, 210
- mroot, 270
  - msolve\_de, 271
  - print\_iter, 271
- msolve\_de
  - mroot, 271
- multi\_funcnt, 272
  - operator(), 272
- multi\_funcnt\_fptr, 273
  - operator(), 273
- multi\_funcnt\_fptr\_noerr, 274
  - operator(), 275
- multi\_funcnt\_gsl, 275
  - operator(), 276
- multi\_funcnt\_mfptr, 276
  - operator(), 277
- multi\_funcnt\_mfptr\_noerr, 277
  - operator(), 278
- multi\_inte, 278
  - get\_error, 279
- multi\_min, 279
  - print\_iter, 280
- multi\_min\_fix, 280
- multi\_vfuncnt, 282
  - operator(), 282
- multi\_vfuncnt\_fptr, 282
  - operator(), 283
- multi\_vfuncnt\_fptr\_noerr, 284
  - operator(), 284
- multi\_vfuncnt\_gsl, 285
  - operator(), 286
- multi\_vfuncnt\_mfptr, 286
  - operator(), 287
- multi\_vfuncnt\_mfptr\_noerr, 287
  - operator(), 288
- naive\_metropolis, 288
  - minteg\_array, 289
  - minteg\_err, 289
- naive\_quartic\_complex, 290
  - solve\_c, 290
- naive\_quartic\_real, 291
  - solve\_r, 291
- new\_column

- table, 384
- nmsimplex\_size
  - gsl\_mmin\_simp\_b, 211
- nonadapt\_step, 292
  - astep, 292
  - astep\_derivs, 292
- npointers
  - coutput, 123
- nseg
  - cern\_adapt, 82
- o2scl\_arith, 58
- o2scl\_const, 59
  - e2\_gaussian, 60
  - e2\_hlorentz, 60
  - e2\_mkasa, 61
- o2scl\_fm, 61
  - mass\_alpha, 62
- o2scl\_inte\_qag\_coeffs, 62
  - qk15\_wg, 63
  - qk15\_wgk, 63
  - qk15\_xgk, 63
  - qk21\_wg, 63
  - qk21\_wgk, 63
  - qk21\_xgk, 63
  - qk31\_wg, 63
  - qk31\_wgk, 63
  - qk31\_xgk, 64
  - qk41\_wg, 64
  - qk41\_wgk, 64
  - qk41\_xgk, 64
  - qk51\_wg, 64
  - qk51\_wgk, 64
  - qk51\_xgk, 64
  - qk61\_wg, 64
  - qk61\_wgk, 64
  - qk61\_xgk, 64
- o2scl\_inte\_qng\_coeffs, 65
  - w10, 65
  - w21a, 65
  - w21b, 65
  - w43a, 65
  - w43b, 65
  - w87a, 66
  - w87b, 66
  - x1, 66
  - x2, 66
  - x3, 66
  - x4, 66
- o2scl\_interp, 293
- o2scl\_interp\_vec, 294
- O2SCL\_OP\_CMAT\_CVEC\_MULT
  - vec\_arith.h, 452
- O2SCL\_OP\_CX\_DOT\_PROD
  - vec\_arith.h, 452
- O2SCL\_OP\_DOT\_PROD
  - vec\_arith.h, 453
- O2SCL\_OP\_MAT\_VEC\_MULT
  - vec\_arith.h, 453
- O2SCL\_OP\_SCA\_VEC\_MULT
  - vec\_arith.h, 453
- O2SCL\_OP\_TRANS\_MULT
  - vec\_arith.h, 453
- O2SCL\_OP\_VEC\_MAT\_MULT
  - vec\_arith.h, 453
- O2SCL\_OP\_VEC\_SCA\_MULT
  - vec\_arith.h, 453
- O2SCL\_OP\_VEC\_VEC\_ADD
  - vec\_arith.h, 453
- O2SCL\_OP\_VEC\_VEC\_PRO
  - vec\_arith.h, 454
- O2SCL\_OP\_VEC\_VEC\_SUB
  - vec\_arith.h, 454
- object\_in\_mem
  - io\_tlate, 245
- ode\_bv\_shoot, 295
- ode\_bv\_solve, 296
- ode\_funct, 297
- ode\_funct\_fptr, 298
- ode\_funct\_mfptr, 299
- ode\_it\_funct, 300
- ode\_it\_funct\_fptr, 300
- ode\_it\_funct\_mfptr, 301
- ode\_it\_make\_Coord, 302
- ode\_it\_solve, 303
- ode\_iv\_solve, 304
  - solve\_final\_value, 305
  - solve\_grid, 305
  - solve\_table, 305
- ode\_vfunct, 305
- ode\_vfunct\_fptr, 306
- ode\_vfunct\_mfptr, 307
- odestep, 308
  - step, 309
- ofmatrix, 309
- ofvector, 309
- ofvector\_cx, 310
- omatrix\_alloc, 311
- omatrix\_col\_tlate, 311
- omatrix\_const\_col\_tlate, 312
- omatrix\_const\_row\_tlate, 312
- omatrix\_cx\_col\_tlate, 313
- omatrix\_cx\_const\_col\_tlate, 313
- omatrix\_cx\_const\_row\_tlate, 314
- omatrix\_cx\_row\_tlate, 314
- omatrix\_cx\_tlate, 315
  - free, 316
  - transpose, 316
- omatrix\_cx\_tlate.h, 435
- omatrix\_cx\_view\_tlate, 316
  - cols, 318

- rows, 318
- tda, 318
- omatrix\_diag\_tlate, 318
- omatrix\_row\_tlate, 319
- omatrix\_tlate, 319
  - free, 320
  - transpose, 320
- omatrix\_tlate.h, 436
  - operator<<, 438
- omatrix\_view\_tlate, 321
  - cols, 322
  - rows, 322
  - tda, 322
- open
  - file\_detect, 140
- operator()
  - multi\_funct, 272
  - multi\_funct\_fptr, 273
  - multi\_funct\_fptr\_noerr, 275
  - multi\_funct\_gsl, 276
  - multi\_funct\_mfptr, 277
  - multi\_funct\_mfptr\_noerr, 278
  - multi\_vfunct, 282
  - multi\_vfunct\_fptr, 283
  - multi\_vfunct\_fptr\_noerr, 284
  - multi\_vfunct\_gsl, 286
  - multi\_vfunct\_mfptr, 287
  - multi\_vfunct\_mfptr\_noerr, 288
- operator<<
  - omatrix\_tlate.h, 438
  - ovector\_cx\_tlate.h, 440
  - ovector\_tlate.h, 442
  - umatrix\_cx\_tlate.h, 446
  - umatrix\_tlate.h, 448
  - uvector\_cx\_tlate.h, 449
  - uvector\_tlate.h, 451
- operator[]
  - ovector\_const\_reverse\_tlate, 329
  - ovector\_const\_subvector\_reverse\_tlate, 330
  - table, 384
- ordered\_interval
  - search\_vec, 368
- ordered\_lookup
  - search\_vec, 367
  - table, 385
- other\_ioc, 323
- other\_todos\_and\_bugs, 323
- out\_file\_format, 324
- out\_one
  - collection, 105
- ovector\_alloc, 325
- ovector\_array\_stride\_tlate, 325
- ovector\_array\_tlate, 326
- ovector\_const\_array\_stride\_tlate, 326
- ovector\_const\_array\_tlate, 327
- ovector\_const\_reverse\_tlate, 328
  - operator[], 329
- ovector\_const\_subvector\_reverse\_tlate, 329
  - operator[], 330
- ovector\_const\_subvector\_tlate, 330
- ovector\_cx\_array\_stride\_tlate, 332
- ovector\_cx\_array\_tlate, 332
- ovector\_cx\_const\_array\_stride\_tlate, 333
- ovector\_cx\_const\_array\_tlate, 334
- ovector\_cx\_const\_subvector\_tlate, 335
- ovector\_cx\_imag\_tlate, 336
- ovector\_cx\_real\_tlate, 336
- ovector\_cx\_subvector\_tlate, 337
- ovector\_cx\_tlate, 337
  - free, 339
- ovector\_cx\_tlate.h, 438
  - operator<<, 440
- ovector\_cx\_view\_tlate, 339
  - size, 341
  - stride, 341
- ovector\_int\_alloc, 341
- ovector\_reverse\_tlate, 342
- ovector\_subvector\_reverse\_tlate, 343
- ovector\_subvector\_tlate, 344
- ovector\_tlate, 344
  - free, 345
  - reserve, 345
- ovector\_tlate.h, 440
  - operator<<, 442
- ovector\_view\_tlate, 346
  - capacity, 348
  - is\_owner, 348
  - lookup, 348
  - size, 348
  - stride, 348
- pinside, 349
  - inside, 350
  - intersect, 349
- pinside::line, 350
- pinside::point, 350
- planar\_intp, 351
  - interp, 352
- pointer\_alloc, 352
- pointer\_input, 352
- pointer\_lookup
  - coutput, 123
- pointer\_out
  - io\_base, 240
- pointer\_output, 353
- poly.h, 442
- poly\_complex, 353
  - solve\_c, 354
- poly\_real\_coeff, 354
  - solve\_rc, 355
- polylog, 355

- print\_iter
    - fit\_base, 141
    - gsl\_mmin\_simp, 210
    - minimize, 259
    - mroot, 271
    - multi\_min, 280
    - root, 366
    - sim\_anneal, 369
  - print\_simplex
    - gsl\_mmin\_simp, 211
  - product
    - lanczos, 252
  - ptos
    - string\_conv.h, 445
  - qk15\_wg
    - o2scl\_inte\_qag\_coeffs, 63
  - qk15\_wgk
    - o2scl\_inte\_qag\_coeffs, 63
  - qk15\_xgk
    - o2scl\_inte\_qag\_coeffs, 63
  - qk21\_wg
    - o2scl\_inte\_qag\_coeffs, 63
  - qk21\_wgk
    - o2scl\_inte\_qag\_coeffs, 63
  - qk21\_xgk
    - o2scl\_inte\_qag\_coeffs, 63
  - qk31\_wg
    - o2scl\_inte\_qag\_coeffs, 63
  - qk31\_wgk
    - o2scl\_inte\_qag\_coeffs, 63
  - qk31\_xgk
    - o2scl\_inte\_qag\_coeffs, 64
  - qk41\_wg
    - o2scl\_inte\_qag\_coeffs, 64
  - qk41\_wgk
    - o2scl\_inte\_qag\_coeffs, 64
  - qk41\_xgk
    - o2scl\_inte\_qag\_coeffs, 64
  - qk51\_wg
    - o2scl\_inte\_qag\_coeffs, 64
  - qk51\_wgk
    - o2scl\_inte\_qag\_coeffs, 64
  - qk51\_xgk
    - o2scl\_inte\_qag\_coeffs, 64
  - qk61\_wg
    - o2scl\_inte\_qag\_coeffs, 64
  - qk61\_wgk
    - o2scl\_inte\_qag\_coeffs, 64
  - qk61\_xgk
    - o2scl\_inte\_qag\_coeffs, 64
  - quad\_intp, 356
    - interp, 358
  - quad\_intp::point, 358
  - quadratic\_complex, 358
  - quadratic\_real, 359
  - quadratic\_real\_coeff, 360
  - quadratic\_std\_complex, 360
  - quartic\_complex, 361
    - solve\_c, 362
    - solve\_r, 362
    - solve\_rc, 362
  - quartic\_real, 362
    - solve\_r, 363
  - quartic\_real\_coeff, 363
    - solve\_r, 363
    - solve\_rc, 364
  - regrid\_data
    - contour, 121
  - remove
    - collection, 105
  - remove\_type
    - io\_type\_info, 247
  - rename\_column
    - table, 385
  - report
    - test\_mgr, 389
  - reserve
    - ovector\_tlate, 345
  - reset\_interp
    - twod\_intp, 399
  - reset\_list
    - table, 387
  - retrieve
    - gsl\_inte\_table, 196
  - rewrite
    - collection, 104
  - rnga, 364
    - clock\_seed, 365
  - root, 365
    - deriv\_eps, 366
    - print\_iter, 366
  - rows
    - omatrix\_cx\_view\_tlate, 318
    - omatrix\_view\_tlate, 322
    - umatrix\_cx\_view\_tlate, 406
    - umatrix\_view\_tlate, 410
  - screenify
    - misc.h, 434
  - search\_vec, 367
    - bsearch\_dec, 368
    - bsearch\_inc, 368
    - ordered\_interval, 368
    - ordered\_lookup, 367
  - set
    - gsl\_mmin\_conf, 205
    - gsl\_root\_brent, 231
    - gsl\_root\_stef, 232
    - table, 383
  - set\_alias
-

- cli, 100
  - set\_comm\_option
    - cli, 100
  - set\_data
    - contour, 121
    - twod\_intp, 398
  - set\_de
    - gsl\_mmin\_conf, 205
    - gsl\_mroot\_hybrids, 217
  - set\_delta
    - cern\_minimize, 89
  - set\_key
    - gsl\_inte\_qag, 178
  - set\_levels
    - contour, 121
  - set\_mode
    - cern\_root, 95
  - set\_nlines
    - table, 383
  - set\_nlines\_auto
    - table, 383
  - set\_npoints
    - eqi\_deriv, 134
  - set\_npoints2
    - eqi\_deriv, 134
  - set\_order
    - gsl\_chebapp, 167
  - set\_output\_level
    - test\_mgr, 389
  - set\_parameters
    - cli, 100
  - set\_ptrs
    - comp\_gen\_inte, 113
    - composite\_inte, 115
  - set\_step
    - adapt\_step, 68
  - set\_type
    - twod\_eqi\_intp, 397
  - set\_verbose
    - cli, 100
  - shrink\_step
    - gsl\_mroot\_hybrids, 217
  - sim\_anneal, 368
    - print\_iter, 369
  - simple\_grad, 370
  - simple\_grad\_array, 370
  - simple\_jacobian, 371
  - size
    - ovector\_cx\_view\_tlate, 341
    - ovector\_view\_tlate, 348
    - uvector\_cx\_view\_tlate, 418
    - uvector\_view\_tlate, 421
  - size\_of\_exponent
    - string\_conv.h, 445
  - sma\_interp, 372
  - sma\_interp\_vec, 373
  - smart\_interp, 373
    - find\_subset, 374
  - smart\_interp\_vec, 375
  - smooth\_contours
    - contour, 122
  - solve\_bkt
    - gsl\_root\_brent, 231
  - solve\_c
    - cubic\_complex, 126
    - cubic\_std\_complex, 128
    - naive\_quartic\_complex, 290
    - poly\_complex, 354
    - quartic\_complex, 362
  - solve\_final\_value
    - ode\_iv\_solve, 305
  - solve\_grid
    - ode\_iv\_solve, 305
  - solve\_r
    - gsl\_quartic\_real, 221
    - gsl\_quartic\_real2, 222
    - naive\_quartic\_real, 291
    - quartic\_complex, 362
    - quartic\_real, 363
    - quartic\_real\_coeff, 363
  - solve\_rc
    - cern\_quartic\_real\_coeff, 94
    - gsl\_poly\_real\_coeff, 219
    - poly\_real\_coeff, 355
    - quartic\_complex, 362
    - quartic\_real\_coeff, 364
  - solve\_table
    - ode\_iv\_solve, 305
  - sortd, 376
  - stat\_input
    - io\_tlate, 245
    - io\_vtlate, 248
  - step
    - gsl\_rk8pd, 224
    - gsl\_rk8pd\_fast, 225
    - gsl\_rkck, 227
    - gsl\_rkck\_fast, 228
    - odestep, 309
  - stod
    - string\_conv.h, 445
  - stoi
    - string\_conv.h, 445
  - stride
    - ovector\_cx\_view\_tlate, 341
    - ovector\_view\_tlate, 348
  - string\_comp, 376
  - string\_conv.h, 443
    - double\_to\_html, 444
    - double\_to\_ieee\_string, 444
    - double\_to\_latex, 444
-

- dtos, 445
- ptos, 445
- size\_of\_exponent, 445
- stod, 445
- stoi, 445
- string\_io\_type, 377
- subtable
  - table, 386
- summary
  - table, 387
- table, 377
  - add\_col\_from\_table, 385
  - ch\_owner, 385
  - clear\_data, 387
  - delete\_column, 387
  - deriv, 386
  - deriv2, 386
  - get\_column, 383, 384
  - init\_column, 385
  - integ, 386
  - interp, 385, 386
  - interp\_const, 386
  - lookup\_column, 385
  - lookup\_form, 385
  - new\_column, 384
  - operator[], 384
  - ordered\_lookup, 385
  - rename\_column, 385
  - reset\_list, 387
  - set, 383
  - set\_nlines, 383
  - set\_nlines\_auto, 383
  - subtable, 386
  - summary, 387
- tda
  - omatrix\_cx\_view\_tlate, 318
  - omatrix\_view\_tlate, 322
- test\_mgr, 387
  - report, 389
  - set\_output\_level, 389
- text\_in\_file, 389
- text\_out\_file, 390
  - text\_out\_file, 393
- timer\_clock, 393
- timer\_gettod, 394
- tptr\_geoseries, 394
- tptr\_schedule, 395
- transpose
  - omatrix\_cx\_tlate, 316
  - omatrix\_tlate, 320
  - umatrix\_cx\_tlate, 404
  - umatrix\_tlate, 408
- twod\_eqi\_intp, 396
  - set\_type, 397
- twod\_intp, 397
  - reset\_interp, 399
  - set\_data, 398
- ufmatrix, 399
- ufmatrix\_cx, 399
- ufvector, 400
- umatrix\_alloc, 401
- umatrix\_const\_row\_tlate, 401
- umatrix\_cx\_alloc, 402
- umatrix\_cx\_const\_row\_tlate, 402
- umatrix\_cx\_row\_tlate, 403
- umatrix\_cx\_tlate, 403
  - free, 404
  - transpose, 404
- umatrix\_cx\_tlate.h, 445
  - operator<<, 446
- umatrix\_cx\_view\_tlate, 405
  - cols, 406
  - rows, 406
- umatrix\_row\_tlate, 407
- umatrix\_tlate, 407
  - free, 408
  - transpose, 408
- umatrix\_tlate.h, 447
  - operator<<, 448
- umatrix\_view\_tlate, 409
  - cols, 410
  - rows, 410
- uvector\_array\_tlate, 411
- uvector\_const\_array\_tlate, 411
- uvector\_const\_subvector\_tlate, 412
- uvector\_cx\_array\_tlate, 413
- uvector\_cx\_const\_array\_tlate, 413
- uvector\_cx\_const\_subvector\_tlate, 414
- uvector\_cx\_subvector\_tlate, 415
- uvector\_cx\_tlate, 415
  - free, 416
- uvector\_cx\_tlate.h, 448
  - operator<<, 449
- uvector\_cx\_view\_tlate, 416
  - size, 418
- uvector\_subvector\_tlate, 418
- uvector\_tlate, 419
  - free, 420
- uvector\_tlate.h, 450
  - operator<<, 451
- uvector\_view\_tlate, 420
  - lookup, 421
  - size, 421
- vec\_arith.h, 451
  - O2SCL\_OP\_CMAT\_CVEC\_MULT, 452
  - O2SCL\_OP\_CX\_DOT\_PROD, 452
  - O2SCL\_OP\_DOT\_PROD, 453
  - O2SCL\_OP\_MAT\_VEC\_MULT, 453
  - O2SCL\_OP\_SCA\_VEC\_MULT, 453

---

- O2SCL\_OP\_TRANS\_MULT, [453](#)
- O2SCL\_OP\_VEC\_MAT\_MULT, [453](#)
- O2SCL\_OP\_VEC\_SCA\_MULT, [453](#)
- O2SCL\_OP\_VEC\_VEC\_ADD, [453](#)
- O2SCL\_OP\_VEC\_VEC\_PRO, [454](#)
- O2SCL\_OP\_VEC\_VEC\_SUB, [454](#)
- vector\_avg
  - array.h, [424](#)
- vector\_out
  - array.h, [425](#)
- vector\_sum
  - array.h, [425](#)
- vector\_variance
  - array.h, [425](#)
- w10
  - o2scl\_inte\_qng\_coeffs, [65](#)
- w21a
  - o2scl\_inte\_qng\_coeffs, [65](#)
- w21b
  - o2scl\_inte\_qng\_coeffs, [65](#)
- w43a
  - o2scl\_inte\_qng\_coeffs, [65](#)
- w43b
  - o2scl\_inte\_qng\_coeffs, [65](#)
- w87a
  - o2scl\_inte\_qng\_coeffs, [66](#)
- w87b
  - o2scl\_inte\_qng\_coeffs, [66](#)
- word\_io\_type, [422](#)
- x1
  - o2scl\_inte\_qng\_coeffs, [66](#)
- x2
  - o2scl\_inte\_qng\_coeffs, [66](#)
- x3
  - o2scl\_inte\_qng\_coeffs, [66](#)
- x4
  - o2scl\_inte\_qng\_coeffs, [66](#)

---