

O₂scl - An Object-Oriented Scientific Computing Library

Version 0.904

Copyright © 2006, 2007, 2008, 2009 Andrew W. Steiner

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 “License Information”.

Contents

1	O2scl User's Guide	1
1.1	Quick Reference to User's Guide	1
1.2	Installation	3
1.3	General Usage	3
1.4	Compiling examples	5
1.5	Related projects	5
1.6	Complex Numbers	6
1.7	Arrays, Vectors, Matrices and Tensors	7
1.8	Permutations	14
1.9	Linear algebra	14
1.10	Interpolation	15
1.11	Physical constants	16
1.12	Function Objects	16
1.13	Data tables	17
1.14	String manipulation	17
1.15	Differentiation	18
1.16	Integration	18
1.17	Roots of Polynomials	19
1.18	Equation Solving	20
1.19	Minimization	20
1.20	Constrained Minimization	21
1.21	Monte Carlo Integration	21
1.22	Simulated Annealing	21
1.23	Non-linear Least-Squares Fitting	21
1.24	Solution of Ordinary Differential Equations	21
1.25	Random Number Generation	22
1.26	Two-dimensional Interpolation	22
1.27	Chebyshev Approximation	22
1.28	Unit Conversions	22
1.29	Other classes and functions	22
1.30	Library settings	23
1.31	Object I/O	23
1.32	Example source code	25
1.33	Design Considerations	48
1.34	License Information	51
1.35	Acknowledgements	67
1.36	Bibliography	67
2	Download O2scl	67
3	Ideas for future development	68
4	Todo List	76
5	Bug List	81
6	Things which need documentation	81
7	Namespace Documentation	81
7.1	gsl_cgs Namespace Reference	81
7.2	gsl_cgsm Namespace Reference	85
7.3	gsl_mks Namespace Reference	89
7.4	gsl_mkxa Namespace Reference	93
7.5	gsl_num Namespace Reference	96
7.6	o2scl Namespace Reference	97

7.7	o2scl_cblas Namespace Reference	97
7.8	o2scl_cblas_paren Namespace Reference	100
7.9	o2scl_const Namespace Reference	100
7.10	o2scl_fm Namespace Reference	102
7.11	o2scl_inte_qag_coeffs Namespace Reference	103
7.12	o2scl_inte_qng_coeffs Namespace Reference	104
7.13	o2scl_linalg Namespace Reference	106
7.14	o2scl_linalg_paren Namespace Reference	112
8	Data Structure Documentation	112
8.1	adapt_step Class Template Reference	112
8.2	akima_interp Class Template Reference	114
8.3	akima_peri_interp Class Template Reference	115
8.4	anneal_mt Class Template Reference	116
8.5	array_2d_alloc Class Template Reference	118
8.6	array_2d_col Class Template Reference	119
8.7	array_2d_row Class Template Reference	119
8.8	array_alloc Class Template Reference	120
8.9	array_const_reverse Class Template Reference	120
8.10	array_const_subvector Class Reference	121
8.11	array_const_subvector_reverse Class Reference	122
8.12	array_interp Class Template Reference	122
8.13	array_interp_vec Class Template Reference	123
8.14	array_reverse Class Template Reference	123
8.15	array_subvector Class Reference	124
8.16	array_subvector_reverse Class Reference	125
8.17	base_interp Class Template Reference	125
8.18	base_interp_mgr Class Template Reference	127
8.19	base_ioc Class Reference	128
8.20	bin_size Class Reference	128
8.21	binary_in_file Class Reference	129
8.22	binary_out_file Class Reference	131
8.23	bool_io_type Class Reference	132
8.24	cern_adapt Class Template Reference	133
8.25	cern_cauchy Class Template Reference	134
8.26	cern_cubic_real_coeff Class Reference	136
8.27	cern_deriv Class Template Reference	137
8.28	cern_gauss Class Template Reference	138
8.29	cern_gauss56 Class Template Reference	140
8.30	cern_minimize Class Template Reference	141
8.31	cern_mroot Class Template Reference	143
8.32	cern_mroot_root Class Template Reference	145
8.33	cern_quartic_real_coeff Class Reference	148
8.34	cern_root Class Template Reference	148
8.35	char_io_type Class Reference	150
8.36	cinput Class Reference	151
8.37	cli Class Reference	152
8.38	cli_readline Class Reference	156
8.39	cmd_line_arg Struct Reference	157
8.40	collection Class Reference	157
8.41	collection::iterator Class Reference	162
8.42	collection::type_iterator Class Reference	163
8.43	collection_entry Struct Reference	164
8.44	columnify Class Reference	164
8.45	comm_option_funct Class Reference	165
8.46	comm_option_mfptr Class Template Reference	166

8.47	comm_option_s Struct Reference	167
8.48	comp_gen_inte Class Template Reference	168
8.49	composite_inte Class Template Reference	169
8.50	contour Class Reference	171
8.51	contour_line Class Reference	175
8.52	convert_units Class Reference	175
8.53	convert_units::unit_t Struct Reference	177
8.54	coutput Class Reference	177
8.55	coutput::ltptr Struct Reference	178
8.56	cspline_interp Class Template Reference	179
8.57	cspline_peri_interp Class Template Reference	180
8.58	cubic_complex Class Reference	181
8.59	cubic_real Class Reference	182
8.60	cubic_real_coeff Class Reference	183
8.61	cubic_std_complex Class Reference	183
8.62	def_interp_mgr Class Template Reference	184
8.63	deriv Class Template Reference	185
8.64	deriv::dpars Struct Reference	187
8.65	double_io_type Class Reference	187
8.66	edge_crossings Class Reference	188
8.67	eqi_deriv Class Template Reference	189
8.68	err_base Class Reference	192
8.69	err_class Class Reference	193
8.70	exact_jacobian Class Template Reference	195
8.71	exact_jacobian::ej_parms Struct Reference	196
8.72	file_detect Class Reference	196
8.73	fit_base Class Template Reference	198
8.74	fit_fix_pars Class Template Reference	199
8.75	fit_funct Class Template Reference	200
8.76	fit_funct_cmfp_ptr Class Template Reference	201
8.77	fit_funct_fp_ptr Class Template Reference	202
8.78	fit_funct_mfp_ptr Class Template Reference	203
8.79	fit_vfunct Class Template Reference	204
8.80	fit_vfunct_cmfp_ptr Class Template Reference	204
8.81	fit_vfunct_fp_ptr Class Template Reference	205
8.82	fit_vfunct_mfp_ptr Class Template Reference	206
8.83	format_float Class Reference	207
8.84	funct Class Template Reference	210
8.85	funct_cmfp_ptr Class Template Reference	211
8.86	funct_cmfp_ptr_noerr Class Template Reference	213
8.87	funct_cmfp_ptr_nopar Class Template Reference	214
8.88	funct_fp_ptr Class Template Reference	215
8.89	funct_fp_ptr_noerr Class Template Reference	215
8.90	funct_fp_ptr_nopar Class Template Reference	216
8.91	funct_mfp_ptr Class Template Reference	217
8.92	funct_mfp_ptr_noerr Class Template Reference	218
8.93	funct_mfp_ptr_nopar Class Template Reference	219
8.94	gaussian_2d Class Template Reference	220
8.95	gen_inte Class Template Reference	221
8.96	gen_test_number Class Template Reference	222
8.97	grad_funct Class Template Reference	223
8.98	grad_funct_cmfp_ptr Class Template Reference	224
8.99	grad_funct_fp_ptr Class Template Reference	225
8.100	grad_funct_mfp_ptr Class Template Reference	226
8.101	grad_vfunct Class Template Reference	226

8.102	grad_vfunct_cmfp Class Template Reference	227
8.103	grad_vfunct_fptr Class Template Reference	228
8.104	grad_vfunct_mfp Class Template Reference	229
8.105	gradient Class Template Reference	229
8.106	gradient_array Class Template Reference	230
8.107	gsl_anneal Class Template Reference	231
8.108	gsl_astep Class Template Reference	233
8.109	gsl_bsimp Class Template Reference	235
8.110	gsl_chebapp Class Reference	238
8.111	gsl_cubic_real_coeff Class Reference	240
8.112	gsl_deriv Class Template Reference	241
8.113	gsl_fft Class Reference	242
8.114	gsl_fit Class Template Reference	243
8.115	gsl_fit::func_par Struct Reference	245
8.116	gsl_inte Class Reference	246
8.117	gsl_inte_cheb Class Template Reference	246
8.118	gsl_inte_kronrod Class Template Reference	248
8.119	gsl_inte_qag Class Template Reference	249
8.120	gsl_inte_qagi Class Template Reference	251
8.121	gsl_inte_qagil Class Template Reference	252
8.122	gsl_inte_qagiu Class Template Reference	254
8.123	gsl_inte_qags Class Template Reference	255
8.124	gsl_inte_qawc Class Template Reference	256
8.125	gsl_inte_qawf_cos Class Template Reference	258
8.126	gsl_inte_qawf_sin Class Template Reference	259
8.127	gsl_inte_qawo_cos Class Template Reference	260
8.128	gsl_inte_qawo_sin Class Template Reference	262
8.129	gsl_inte_qaws Class Template Reference	263
8.130	gsl_inte_qng Class Template Reference	265
8.131	gsl_inte_singular Class Template Reference	266
8.132	gsl_inte_singular::extrapolation_table Struct Reference	268
8.133	gsl_inte_table Class Reference	269
8.134	gsl_inte_transform Class Template Reference	271
8.135	gsl_min_brent Class Template Reference	273
8.136	gsl_miser Class Template Reference	274
8.137	gsl_mmin_base Class Template Reference	278
8.138	gsl_mmin_bfgs2 Class Template Reference	280
8.139	gsl_mmin_bfgs2_array Class Template Reference	282
8.140	gsl_mmin_conf Class Template Reference	283
8.141	gsl_mmin_conf_array Class Template Reference	285
8.142	gsl_mmin_conp Class Template Reference	286
8.143	gsl_mmin_conp_array Class Template Reference	287
8.144	gsl_mmin_linmin Class Reference	288
8.145	gsl_mmin_simp Class Template Reference	289
8.146	gsl_mmin_simp2 Class Template Reference	292
8.147	gsl_mmin_wrap_base Class Reference	295
8.148	gsl_mmin_wrapper Class Template Reference	296
8.149	gsl_monte Class Template Reference	298
8.150	gsl_mroot_hybrids Class Template Reference	299
8.151	gsl_ode_control Class Template Reference	302
8.152	gsl_poly_real_coeff Class Reference	303
8.153	gsl_quadratic_real_coeff Class Reference	304
8.154	gsl_quartic_real Class Reference	305
8.155	gsl_quartic_real2 Class Reference	306
8.156	gsl_rk8pd Class Template Reference	307

8.157	gsl_rk8pd_fast Class Template Reference	308
8.158	gsl_rkck Class Template Reference	310
8.159	gsl_rkck_fast Class Template Reference	311
8.160	gsl_rkf45 Class Template Reference	313
8.161	gsl_rkf45_fast Class Template Reference	314
8.162	gsl_rnga Class Reference	316
8.163	gsl_root_brent Class Template Reference	317
8.164	gsl_root_stef Class Template Reference	319
8.165	gsl_series Class Reference	321
8.166	gsl_solver_HH Class Reference	321
8.167	gsl_solver_LU Class Reference	322
8.168	gsl_solver_QR Class Reference	322
8.169	gsl_vegas Class Template Reference	323
8.170	hybrids_base Class Reference	327
8.171	in_file_format Class Reference	328
8.172	int_io_type Class Reference	329
8.173	inte Class Template Reference	330
8.174	io_base Class Reference	331
8.175	io_manager Class Reference	333
8.176	io_tlate Class Template Reference	335
8.177	io_type_info Class Reference	338
8.178	io_vtlate Class Template Reference	339
8.179	jac_funct Class Template Reference	341
8.180	jac_funct_cmfprr Class Template Reference	341
8.181	jac_funct_fptr Class Template Reference	342
8.182	jac_funct_mfprr Class Template Reference	343
8.183	jac_vfunrr Class Template Reference	344
8.184	jac_vfunrr_cmfprr Class Template Reference	344
8.185	jac_vfunrr_fptr Class Template Reference	345
8.186	jac_vfunrr_mfprr Class Template Reference	346
8.187	jacobian Class Template Reference	346
8.188	lanczos Class Template Reference	347
8.189	lib_settings_class Class Reference	349
8.190	linear_interp Class Template Reference	350
8.191	linear_solver Class Template Reference	350
8.192	linear_solver_hh Class Template Reference	351
8.193	linear_solver_lu Class Template Reference	352
8.194	linear_solver_qr Class Template Reference	352
8.195	long_io_type Class Reference	353
8.196	mcarlo_inte Class Template Reference	354
8.197	min_fit Class Template Reference	354
8.198	min_fit::func_par Struct Reference	356
8.199	minimize Class Template Reference	357
8.200	minimize_bkt Class Template Reference	359
8.201	minimize_de Class Template Reference	360
8.202	mm_funct Class Template Reference	362
8.203	mm_funct_cmfprr Class Template Reference	362
8.204	mm_funct_cmfprr_nopar Class Template Reference	363
8.205	mm_funct_fptr Class Template Reference	364
8.206	mm_funct_fptr_nopar Class Template Reference	365
8.207	mm_funct_gsl Class Template Reference	366
8.208	mm_funct_mfprr Class Template Reference	367
8.209	mm_funct_mfprr_nopar Class Template Reference	368
8.210	mm_vfunrr Class Template Reference	369
8.211	mm_vfunrr_cmfprr Class Template Reference	369

8.212	mm_vfunct_cmfp_ptr_nopar Class Template Reference	370
8.213	mm_vfunct_fptr Class Template Reference	371
8.214	mm_vfunct_fptr_nopar Class Template Reference	372
8.215	mm_vfunct_gsl Class Template Reference	373
8.216	mm_vfunct_mfp_ptr Class Template Reference	373
8.217	mm_vfunct_mfp_ptr_nopar Class Template Reference	374
8.218	mroot Class Template Reference	375
8.219	multi_funct Class Template Reference	377
8.220	multi_funct_cmfp_ptr Class Template Reference	378
8.221	multi_funct_cmfp_ptr_noerr Class Template Reference	379
8.222	multi_funct_fptr Class Template Reference	380
8.223	multi_funct_fptr_noerr Class Template Reference	381
8.224	multi_funct_gsl Class Template Reference	382
8.225	multi_funct_mfp_ptr Class Template Reference	383
8.226	multi_funct_mfp_ptr_noerr Class Template Reference	384
8.227	multi_inte Class Template Reference	386
8.228	multi_min Class Template Reference	387
8.229	multi_min_fix Class Template Reference	389
8.230	multi_vfunct Class Template Reference	390
8.231	multi_vfunct_cmfp_ptr Class Template Reference	391
8.232	multi_vfunct_cmfp_ptr_noerr Class Template Reference	392
8.233	multi_vfunct_fptr Class Template Reference	393
8.234	multi_vfunct_fptr_noerr Class Template Reference	394
8.235	multi_vfunct_gsl Class Template Reference	395
8.236	multi_vfunct_mfp_ptr Class Template Reference	396
8.237	multi_vfunct_mfp_ptr_noerr Class Template Reference	397
8.238	nonadapt_step Class Template Reference	399
8.239	o2scl_hybrid_state_t Class Template Reference	400
8.240	o2scl_interp Class Template Reference	402
8.241	o2scl_interp_vec Class Template Reference	403
8.242	ode_bv_multishoot Class Template Reference	405
8.243	ode_bv_shoot Class Template Reference	406
8.244	ode_bv_solve Class Template Reference	408
8.245	ode_funct Class Template Reference	409
8.246	ode_funct_cmfp_ptr Class Template Reference	409
8.247	ode_funct_fptr Class Template Reference	410
8.248	ode_funct_mfp_ptr Class Template Reference	411
8.249	ode_it_funct Class Template Reference	412
8.250	ode_it_funct_fptr Class Template Reference	412
8.251	ode_it_funct_mfp_ptr Class Template Reference	413
8.252	ode_it_solve Class Template Reference	414
8.253	ode_iv_solve Class Template Reference	415
8.254	ode_jac_funct Class Template Reference	418
8.255	ode_jac_funct_cmfp_ptr Class Template Reference	419
8.256	ode_jac_funct_fptr Class Template Reference	420
8.257	ode_jac_funct_mfp_ptr Class Template Reference	421
8.258	ode_jac_vfunct Class Template Reference	422
8.259	ode_jac_vfunct_cmfp_ptr Class Template Reference	423
8.260	ode_jac_vfunct_fptr Class Template Reference	423
8.261	ode_jac_vfunct_mfp_ptr Class Template Reference	424
8.262	ode_vfunct Class Template Reference	425
8.263	ode_vfunct_cmfp_ptr Class Template Reference	426
8.264	ode_vfunct_fptr Class Template Reference	427
8.265	ode_vfunct_mfp_ptr Class Template Reference	428
8.266	odestep Class Template Reference	428

8.267	ofvector Class Template Reference	429
8.268	ofvector_cx Class Template Reference	430
8.269	omatrix_alloc Class Reference	431
8.270	omatrix_array_tlate Class Template Reference	431
8.271	omatrix_base_tlate Class Template Reference	432
8.272	omatrix_col_tlate Class Template Reference	433
8.273	omatrix_const_col_tlate Class Template Reference	434
8.274	omatrix_const_diag_tlate Class Template Reference	435
8.275	omatrix_const_row_tlate Class Template Reference	435
8.276	omatrix_const_view_tlate Class Template Reference	436
8.277	omatrix_cx_col_tlate Class Template Reference	437
8.278	omatrix_cx_const_col_tlate Class Template Reference	438
8.279	omatrix_cx_const_row_tlate Class Template Reference	439
8.280	omatrix_cx_row_tlate Class Template Reference	439
8.281	omatrix_cx_tlate Class Template Reference	440
8.282	omatrix_cx_view_tlate Class Template Reference	441
8.283	omatrix_diag_tlate Class Template Reference	443
8.284	omatrix_row_tlate Class Template Reference	443
8.285	omatrix_tlate Class Template Reference	444
8.286	omatrix_view_tlate Class Template Reference	446
8.287	ool_constr_mmin Class Template Reference	447
8.288	ool_hfunct Class Template Reference	450
8.289	ool_hfunct_fptr Class Template Reference	450
8.290	ool_hfunct_mfptr Class Template Reference	451
8.291	ool_hvfunct Class Template Reference	452
8.292	ool_hvfunct_fptr Class Template Reference	452
8.293	ool_hvfunct_mfptr Class Template Reference	453
8.294	ool_mmin_gencan Class Template Reference	454
8.295	ool_mmin_pgrad Class Template Reference	457
8.296	ool_mmin_spg Class Template Reference	459
8.297	other_todos_and_bugs Class Reference	460
8.298	out_file_format Class Reference	462
8.299	ovector_alloc Class Reference	463
8.300	ovector_array_stride_tlate Class Template Reference	463
8.301	ovector_array_tlate Class Template Reference	464
8.302	ovector_base_tlate Class Template Reference	464
8.303	ovector_const_array_stride_tlate Class Template Reference	467
8.304	ovector_const_array_tlate Class Template Reference	467
8.305	ovector_const_reverse_tlate Class Template Reference	468
8.306	ovector_const_subvector_reverse_tlate Class Template Reference	469
8.307	ovector_const_subvector_tlate Class Template Reference	470
8.308	ovector_const_view_tlate Class Template Reference	470
8.309	ovector_cx_array_stride_tlate Class Template Reference	474
8.310	ovector_cx_array_tlate Class Template Reference	474
8.311	ovector_cx_const_array_stride_tlate Class Template Reference	475
8.312	ovector_cx_const_array_tlate Class Template Reference	476
8.313	ovector_cx_const_subvector_tlate Class Template Reference	477
8.314	ovector_cx_imag_tlate Class Template Reference	478
8.315	ovector_cx_real_tlate Class Template Reference	479
8.316	ovector_cx_subvector_tlate Class Template Reference	479
8.317	ovector_cx_tlate Class Template Reference	480
8.318	ovector_cx_view_tlate Class Template Reference	481
8.319	ovector_int_alloc Class Reference	484
8.320	ovector_reverse_tlate Class Template Reference	484
8.321	ovector_subvector_reverse_tlate Class Template Reference	486

8.322	ovector_subvector_tlate Class Template Reference	487
8.323	ovector_tlate Class Template Reference	487
8.324	ovector_view_tlate Class Template Reference	490
8.325	permutation Class Reference	492
8.326	pinside Class Reference	493
8.327	pinside::line Struct Reference	495
8.328	pinside::point Struct Reference	495
8.329	planar_intp Class Template Reference	495
8.330	pointer_2d_alloc Class Template Reference	498
8.331	pointer_alloc Class Template Reference	498
8.332	pointer_input Struct Reference	499
8.333	pointer_output Struct Reference	499
8.334	poly_complex Class Reference	500
8.335	poly_real_coeff Class Reference	500
8.336	polylog Class Reference	501
8.337	quadratic_complex Class Reference	503
8.338	quadratic_real Class Reference	503
8.339	quadratic_real_coeff Class Reference	504
8.340	quadratic_std_complex Class Reference	505
8.341	quartic_complex Class Reference	505
8.342	quartic_real Class Reference	506
8.343	quartic_real_coeff Class Reference	507
8.344	rnga Class Reference	508
8.345	root Class Template Reference	509
8.346	root_bkt Class Template Reference	510
8.347	root_de Class Template Reference	511
8.348	search_vec Class Template Reference	512
8.349	sim_anneal Class Template Reference	515
8.350	simple_grad Class Template Reference	516
8.351	simple_grad_array Class Template Reference	517
8.352	simple_jacobian Class Template Reference	517
8.353	simple_quartic_complex Class Reference	519
8.354	simple_quartic_real Class Reference	519
8.355	sma_interp Class Template Reference	520
8.356	sma_interp_vec Class Template Reference	521
8.357	smart_interp Class Template Reference	521
8.358	smart_interp_vec Class Template Reference	523
8.359	string_comp Struct Reference	525
8.360	string_io_type Class Reference	526
8.361	table Class Reference	526
8.362	table3d Class Reference	538
8.363	table::col_s Struct Reference	543
8.364	table::sortd_s Struct Reference	544
8.365	table_units Class Reference	544
8.366	tensor Class Reference	546
8.367	tensor1 Class Reference	548
8.368	tensor2 Class Reference	549
8.369	tensor3 Class Reference	550
8.370	tensor4 Class Reference	550
8.371	tensor_grid Class Reference	551
8.372	tensor_grid2 Class Reference	553
8.373	tensor_grid3 Class Reference	554
8.374	tensor_grid4 Class Reference	555
8.375	test_mgr Class Reference	556
8.376	text_in_file Class Reference	558

8.377	text_out_file Class Reference	559
8.378	twod_eqi_intp Class Reference	562
8.379	twod_intp Class Reference	563
8.380	ufmatrix Class Template Reference	565
8.381	ufmatrix_cx Class Template Reference	565
8.382	ufvector Class Template Reference	566
8.383	umatrix_alloc Class Reference	566
8.384	umatrix_base_tlate Class Template Reference	567
8.385	umatrix_const_row_tlate Class Template Reference	568
8.386	umatrix_const_view_tlate Class Template Reference	569
8.387	umatrix_cx_alloc Class Reference	570
8.388	umatrix_cx_const_row_tlate Class Template Reference	571
8.389	umatrix_cx_row_tlate Class Template Reference	571
8.390	umatrix_cx_tlate Class Template Reference	572
8.391	umatrix_cx_view_tlate Class Template Reference	573
8.392	umatrix_row_tlate Class Template Reference	575
8.393	umatrix_tlate Class Template Reference	576
8.394	umatrix_view_tlate Class Template Reference	577
8.395	uvector_alloc Class Reference	578
8.396	uvector_array_tlate Class Template Reference	579
8.397	uvector_base_tlate Class Template Reference	579
8.398	uvector_const_array_tlate Class Template Reference	581
8.399	uvector_const_subvector_tlate Class Template Reference	581
8.400	uvector_const_view_tlate Class Template Reference	582
8.401	uvector_cx_array_tlate Class Template Reference	584
8.402	uvector_cx_const_array_tlate Class Template Reference	584
8.403	uvector_cx_const_subvector_tlate Class Template Reference	585
8.404	uvector_cx_subvector_tlate Class Template Reference	586
8.405	uvector_cx_tlate Class Template Reference	586
8.406	uvector_cx_view_tlate Class Template Reference	587
8.407	uvector_int_alloc Class Reference	589
8.408	uvector_subvector_tlate Class Template Reference	589
8.409	uvector_tlate Class Template Reference	590
8.410	uvector_view_tlate Class Template Reference	591
8.411	word_io_type Class Reference	593
9	File Documentation	593
9.1	array.h File Reference	593
9.2	cblas_base.h File Reference	596
9.3	collection.h File Reference	597
9.4	columnify.h File Reference	600
9.5	cx_arith.h File Reference	601
9.6	err_hnd.h File Reference	603
9.7	givens.h File Reference	607
9.8	givens_base.h File Reference	607
9.9	hh_base.h File Reference	608
9.10	householder_base.h File Reference	608
9.11	lib_settings.h File Reference	609
9.12	lu_base.h File Reference	610
9.13	minimize.h File Reference	611
9.14	misc.h File Reference	613
9.15	omatrix_cx_tlate.h File Reference	615
9.16	omatrix_tlate.h File Reference	616
9.17	ovector_cx_tlate.h File Reference	618
9.18	ovector_rev_tlate.h File Reference	620
9.19	ovector_tlate.h File Reference	621

9.20	permutation.h File Reference	623
9.21	poly.h File Reference	624
9.22	qr_base.h File Reference	626
9.23	string_conv.h File Reference	626
9.24	tensor.h File Reference	628
9.25	tridiag_base.h File Reference	629
9.26	umatrix_cx_tlate.h File Reference	630
9.27	umatrix_tlate.h File Reference	631
9.28	user_io.h File Reference	633
9.29	uvector_cx_tlate.h File Reference	637
9.30	uvector_tlate.h File Reference	638
9.31	vec_arith.h File Reference	640
9.32	vec_stats.h File Reference	649
9.33	vector.h File Reference	651

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.
- 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

This is a beta version. The library should install and test successfully, and most of the classes are ready for production use. Some of the interfaces may change slightly in future versions. There are a few classes which are more experimental, and this is clearly stated at the top of the documentation for these classes.

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

1.1 Quick Reference to User's Guide

- [Installation](#)
- [General Usage](#)
- [Compiling examples](#)
- [Related projects](#)
- [Complex Numbers](#)

- [Arrays, Vectors, Matrices and Tensors](#)
 - [Permutations](#)
 - [Linear algebra](#)
 - [Interpolation](#)
 - [Physical constants](#)
 - [Function Objects](#)
 - [Data tables](#)
 - [String manipulation](#)
 - [Differentiation](#)
 - [Integration](#)
 - [Roots of Polynomials](#)
 - [Equation Solving](#)
 - [Minimization](#)
 - [Constrained Minimization](#)
 - [Monte Carlo Integration](#)
 - [Simulated Annealing](#)
 - [Non-linear Least-Squares Fitting](#)
 - [Solution of Ordinary Differential Equations](#)
 - [Random Number Generation](#)
 - [Two-dimensional Interpolation](#)
 - [Chebyshev Approximation](#)
 - [Unit Conversions](#)
 - [Other classes and functions](#)
 - [Library settings](#)
 - [Object I/O](#)
 - [Example source code](#)
 - [Design Considerations](#)
 - [License Information](#)
 - [Acknowledgements](#)
 - [Bibliography](#)
 - [Todo List](#)
 - [Bug List](#)
-
-

1.2 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`. More information on the `configure` command can also be obtained from `./configure --help`. `O2scl` requires the GSL libraries. If the `configure` script cannot find them, you may have to specify their location in the `CPPFLAGS` and `LDFLAGS` environment variables (`./configure --help` shows some information on this). The documentation is automatically installed by `make install`.

After `make install`, you may test the library with `make o2scl-test`. At the moment, testing `O2scl` requires that you have `grep`, `awk`, `tail`, `cat`, and `wc` on your machine to summarize the test results, but this will hopefully be improved in later version.

This library requires GSL and is designed to work with GSL versions 1.12 or greater. 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 on by default. You can disable range-checking by defining `-DO2SCL_NO_RANGE_CHECK`

```
CPPFLAGS="-DO2SCL_NO_RANGE_CHECK" ./configure
```

The separate libraries `O2scl_eos` and `O2scl_part` are installed by default. To disable the installation of these libraries and their associated documentation, run `./configure` with the flags `--disable-eoslib` or `--disable-partlib`. Note that `O2scl_eos` depends on `O2scl_part` so using `--disable-partlib` without `--disable-eoslib` will not work.

There are several warning flags that are useful when configuring and compiling with `O2scl`. See the GSL documentation for an excellent discussion, and also see the generic installation documentation in the file `INSTALL` in the `O2scl` top-level directory. For running `configure`, for example, if you do not have privileges to write to `/usr/local`,

```
CPPFLAGS="-O3" -I/home/asteiner/install/include" \
LDFLAGS="-L/home/asteiner/install/lib" ./configure -C \
--prefix=/home/asteiner/install
```

In this example, specifying `-I/home/asteiner/install/include` and `-L/home/asteiner/install/lib` above ensures that the GSL libraries can be found (this is where they are installed on my machine). The `--prefix=/home/asteiner/install` argument to `./configure` ensures that `O2scl` is installed there as well.

The documentation is generated with Doxygen. In principle, the documentation can be regenerated by the end-user, but this is not supported and requires several external applications not included in the distribution.

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`.

1.3 General Usage

1.3.1 Namespaces

Most of the classes reside in the namespace `o2scl` (this namespace has been removed from the documentation for clarity). Numerical constants (many of them based on the GSL constants) are placed in separate namespaces (`gsl_cgs`, `gsl_cgsm`, `gsl_mks`, `gsl_mkxa`, `gsl_num`, `o2scl_const`, and `o2scl_fm`). There are also two namespaces which hold integration coefficients, `o2scl_inte_qag_coeffs` and `o2scl_inte_qng_coeffs`.

1.3.2 Documentation conventions

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

1.3.3 Nomenclature

Classes directly derived from the GNU Scientific Library are preceded by the prefix `gsl_` and classes derived from CERNLIB are preceded by the prefix `cern_`. Some of those classes derived from GSL and CERNLIB operate slightly differently from the original versions. The differences are detailed in the corresponding class documentation.

1.3.4 Basic error handling

Error handling is similar to GSL. When an error occurs, functions and/or classes call a GSL-like error handler and (when appropriate) return a non-zero value. When functions succeed they return 0 (`gsl_success`). The error handler, `err_hnd` is a global pointer to an object of type `err_base`. There is a global default error handler of type `err_class`. The list of error codes is given in the documentation for the file `err_hnd.h`. The default error handler can be replaced by simply assigning the address of a descendant of `err_base` to `err_hnd`. Most of the time, if a function returns an integer value, and the returned value is nonzero, then the error handler was called (at least once). Functions can return success even when the error handler was called during execution, and this happens particularly in template classes where templated methods are beyond the direct control of the function using them. Also, `O2scl` functions never reset the error handler.

The default behavior for all errors is to store the error information, print the information to `cout` and call `exit(error_number)`. You can modify the behavior of the default error handler with the `err_class::set_mode()` function, Mode '2' is the default, mode '1' just prints the information, and mode '0' ignores the error.

Although `O2scl` does not yet have this functionality by default, it is straightforward to define an error handler which throws a C++ exception. Also, object destructors do not generally call the error handler. Internally, `O2scl` does not use `try` blocks, but these can easily be effectively employed by an `O2scl` user.

Errors can be set by the user through the macros `O2SCL_ERR`, which sets an error, and `O2SCL_ERR_RET`, which sets an error and returns the error number.

Functionality similar to `assert()` is provided with the macro `O2SCL_ASSERT`, which exits if its argument is non-zero, and `O2SCL_BOOL_ASSERT` which exits if its argument is false.

Note that the GSL function `gsl_set_error_handler()` is called in the abstract base class `err_base` to set the GSL error handler any time a new instance of `err_base` is created. If the user creates a new handler, that new handler will be used by GSL functions as well.

1.3.5 What is an error?

`O2scl` assumes that errors are events which should happen infrequently. Error handling strategies are often time-consuming and they are not a replacement for normal code flow. However, even with this in mind, one can still distinguish a large spectrum of possibilities from "fatal" errors, those likely to corrupt the stack and/or cause a dreaded "segmentation fault" and "non-fatal" errors, those errors which might cause incorrect results, but might be somehow recoverable. One of the purposes of error handling is to decide if and how these different types of errors should be handled differently. Most errors in `O2scl` result in a call to the error handler by default. Some of the classes which attempt to reach numerical convergence have an option (e.g. `mroot::err_nonconv`) to turn this default behavior off for convergence errors (which are not necessarily "fatal" errors). To set these "convergence" errors, the macros `O2SCL_CONV` and `O2SCL_CONV_RET` can be used.

Another related issue is that `O2scl` often calls functions which are supplied by the user, these user-designed functions may create errors, and the library needs to decide how to deal with them, even though it knows little about what is actually happening inside these user-defined functions. Most of the time, `O2scl` assumes that if a function returns a nonzero value, then an error has occurred and the present calculation should abort.

1.3.6 Objects and scope

O₂scl objects frequently take inputs which are of the form of a reference to a smaller object. This is particularly convenient because it allows a lot of flexibility, while providing a certain degree of safety. However, the user retains the responsibility of ensuring that input objects do not go out of scope before they are utilized by objects which require them. (This is actually no different than the requirements on the user imposed by GSL, for example.)

A simple example of this is provided by the simulated annealing class `gsl_anneal`. Simulated annealing works by providing a temperature annealing schedule, which slowly decreases the temperature. However, if the temperature schedule object goes out of scope before the annealing is done, a segmentation fault will result. For example, the following code will fail:

```
// How not to provide subobjects to O2scl classes

void set_schedule(gsl_anneal<int,multi_func<int> > &ga) {
    tptr_geoseries<ovector_view> tsch;
    tsch.set_series(1.0,1.0e-3,1.1);
    ga.set_tptr_schedule(tsch);
}

void function() {

    gsl_anneal<int,multi_func<int> > ga;
    set_schedule(ga);
    ga.mmin(1,init,result,param,func);
}
```

This will fail because the temperature schedule object goes out of scope (it's a local variable in the function and its destructor is called before the `set_schedule()` function exits) before the `mmin()` function is called.

1.4 Compiling examples

The example programs are in the `examples` directory. After installation, they can be compiled and executed by running `make o2scl-examples` in that directory. This will also test the output of the examples to make sure it is correct and summarize these tests in `examples-summary.txt`. The output for each example is placed in the corresponding file with a `.scr` extension.

Alternatively, you can make the executable for each example in the `examples` directory individually using, e.g. `make ex_mroot`.

See [Example source code](#) for the documented source code of the individual examples

Also, the testing code for each class is sometimes useful for providing examples of their usage. The testing source code for each source file is named with an `_ts.cpp` prefix in the same directory as the class source.

1.5 Related projects

Several noteworthy related projects:

- **GSL - The GNU Scientific Library**

The first truly free, ANSI-compliant, fully tested, and well-documented scientific computing library. The GSL is located at <http://www.gnu.org/software/gsl>. Manual is available at http://www.gnu.org/software/gsl/manual/html_node/. Many GSL routines are included and reworked in O₂scl (the corresponding classes begin with the `gsl_` prefix) and O₂scl was specifically designed to be used with GSL. GSL is a must-have for most serious numerical work in C or C++ and is required for installation of O₂scl.

- **CERNLIB - The gold standard in FORTRAN computing libraries**

Several CERNLIB routines are rewritten completely in C++ and included in O₂scl (they begin with the `cern_` prefix). CERNLIB is located at <http://cernlib.web.cern.ch/cernlib/mathlib.html>

- LAPACK and ATLAS - The gold standard for linear algebra
Available at <http://www.netlib.org/lapack> and <http://www.netlib.org/atlas> . See also <http://www.netlib.org/clapack> .
- QUADPACK - FORTRAN adaptive integration library
This is the library on which the GSL integration routines are based (prefix `gsl_inte_`). It is available at <http://www.netlib.org/quadpack> .
- TNT - Template numerical toolkit (<http://math.nist.gov>)
TNT provides vector and matrix types and basic arithmetic operations. Most of the classes in `O2scl` which use vector and matrix types are templated to allow compatibility with TNT. (Though there are a few small differences.) This software is in the public domain.
- Blitz++ - <http://www.oonumerics.org/blitz>
Another linear algebra library designed in C++ which includes vector and matrix types and basic arithmetic. As the name implies, Blitz++ has the capability to be particularly fast. Distributed with a Perl-like artistic license "or the GPL".
- Boost - <http://www.boost.org>
Free (license is compatible with GPL) peer-reviewed portable C++ source libraries that work well with the C++ Standard Library. Boost also contains `uBlas`, for linear algebra computing. For examples of how to use `O2scl` with `uBlas` vector and matrices, see [uBlas vector example](#).
- FLENS - <http://flens.sourceforge.net>
A C++ library with object-oriented linear algebra types and an interface to BLAS and LAPACK.
- MESA - Modules for Experiments in Stellar Astrophysics (<http://mesa.sourceforge.net>)
An excellent FORTRAN library with accurate low-density equations of state, interpolation, opacities and other routines useful in stellar physics. Work is currently under way to rewrite some of the MESA routines in C/C++ for `O2scl` . Licensed with LGPL (not GPL).
- OOL - Open Optimization Library (<http://ool.sourceforge.net>)
Constrained minimization library designed with GSL in mind. The `O2scl` constrained minimization classes are derived from this library. See [Constrained Minimization](#) .
- Root - CERN's new C++ analysis package (<http://root.cern.ch>)
A gargantuan library for data analysis, focused mostly on high-energy physics. Their histograms, graphics, file I/O and support for large data sets is particularly good.

1.6 Complex Numbers

Several arithmetic operations for `gsl_complex` are defined in [cx_arith.h](#), but no constructor has been written. The object `gsl_complex` is still a `struct`, not a `class`. For example,

```
#include <o2scl/cx_arith.h>
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 also provided in [cx_arith.h](#).

1.7 Arrays, Vectors, Matrices and Tensors

1.7.1 Introduction

The O₂scl library uses a standard nomenclature to distinguish a couple different concepts. The word "array" is always used to refer to C-style arrays, i.e. `double[]`. If there are two dimensions in the array, it is a "two-dimensional array", i.e. `double[][]`. The word "vector" is reserved generic objects with array-like semantics, i.e. any type of object or class which can be treated similar to an array in that it has an function `operator[]` which gives array-like indexing. Thus arrays are vectors, but not all vectors are arrays. There are a couple specific vector types defined in O₂scl like `ovector` and `uvector`. The STL vector `std::vector<double>` is also a vector type in this language. The word "matrix" is reserved for the a generic object which has matrix-like semantics and can be accessed using either `operator()` or two successive applications of `operator[]` (or sometimes both). The O₂scl objects `omatrix` and `umatrix` are matrix objects, as is a C-style two-dimensional array, `double[][]`. The header files are named in this spirit also: functions and classes which operate on generic vector types are in `vector.h` and functions and classes which work only with C-style arrays are in `array.h`. The word "tensor" is used for a generic object which has rank n and then has n associated indices. A vector is just a tensor of rank 1 and a matrix is just a tensor of rank 2.

Most of the classes in O₂scl which use vectors and/or matrices are designed so that they can be used with any appropriately-defined vector or matrix types. This is a major part of the design goals for O₂scl and most of the classes are compatible with matrix and vector objects from GSL, TNT, MV++, uBlas, and Blitz++.

The first index of a matrix type is defined always to be the index associated with "rows" and the second is associated with "columns". The O₂scl matrix output functions respect this notation as well, so that all of the elements of the first row is sent to the screen, then all of the elements in the second row, and so on. With this in mind, one can make the distinction between "row-major" and "column-major" matrix storage. C-style two-dimensional arrays are "row-major" in that the elements of the first row occupy the first locations in memory as opposed "column-major" where the first column occupies the first locations in memory. It is important to note that the majority of the classes in O₂scl do not care about the details of the underlying memory structure, so long as two successive applications of `operator[]` (or in some cases `operator(,)`) works properly. The storage format used by `omatrix` and `umatrix` is row-major, and there are no column-major matrix classes in O₂scl (yet).

1.7.2 Matrix indexing with `[][]` vs. `(,)`

While vector indexing with `operator[]` is very compatible with almost any vector type, matrix indexing is a bit more difficult. There are two options: assume matrix objects provide an `operator[]` method which can be applied twice, i.e. `m[i][j]`, or assume that matrix elements should be referred to with `m(i, j)`. Most of the O₂scl classes use the former approach so that they are also compatible with two-dimensional arrays. However, there are sometimes good reasons to want to use `operator()` for matrix-intensive operations from linear algebra. For this reason, some of the functions given in the `linalg` directory are specified in two forms: the first default form which assumes `[][]`, and the second form with the same name, but in a namespace which has a suffix `_paren` and assumes matrix types use `(,)`.

1.7.3 Rows and columns vs. x and y

Sometimes its useful to think about the rows and columns in a matrix as referring to a elements of a grid, and the matrix indices refer to points in a grid in (x, y) . It might seem intuitive to think of a matrix as `A[ix][iy]` where `ix` and `iy` are the x and y indices because the ordering of the indices is alphabetical. However, it is useful to note that because functions like `matrix_out` print the first "row" first rather than the first column, a matrix constructed as `A[ix][iy]` will be printed out with x on the "vertical axis" and y on the "horizontal axis", so it is sometimes useful to store data in the form `A[iy][ix]` (for example, in the two dimensional interpolation class, `twod_intp`). In any case, all classes which take matrix data as input will document how the matrix ought to be arranged.

1.7.4 Generic vector functions

There are a couple functions which operate on generic vectors of any type in `vector.h`. They perform sorting, summing, rotating, copying, and computations of minima and maxima. For more statistically-oriented operations, see also `vec_stats.h`.

1.7.5 Native matrix and vector types

The rest of this section in the User's guide is dedicated to describing the O₂scl implementations of vector, matrix, and tensor types.

Vectors and matrices are designed using the templates `ovector_tlate` and `omatrix_tlate`, which are compatible with `gsl_vector` and `gsl_matrix`. Vectors and matrices with unit stride are provided in `uvector_tlate` and `umatrix_tlate`. The most commonly used double-precision versions of these template classes are `ovector`, `omatrix`, `uvector` and `umatrix`, and their associated "views" (analogous to GSL vector and matrix views) which are named with a `_view` suffix. The classes `ovector_tlate` and `omatrix_tlate` 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`, and the complex vector and matrix objects `ovector_cx`, `omatrix_cx`, `uvector_cx`, and `umatrix_cx` are straightforward.

1.7.6 Vector and matrix 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 C-style arrays or pointer arrays (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.

1.7.7 Vector and matrix 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`.

1.7.8 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`.

1.7.9 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 recently allocated memory to zero. You can use `ovector::set_all()` with a zero argument to clear a vector (and similarly for a 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.

Although memory allocation in O₂scl vectors is very similar to that in GSL, the user must not mix allocation and deallocation between GSL and O₂scl.

1.7.10 Get and set

Vectors and matrices 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` or an `omatrix` at the same time, then use `ovector::set_all()` or `omatrix::set_all()`.

1.7.11 Range checking

Range checking is performed depending on whether or not `O2SCL_NO_RANGE_CHECK` is defined. It can be defined in the arguments to `./configure` upon installation to turn off range checking. Note that this is completely separate from the GSL range checking mechanism, so range checking may be on in `O2scl` even if it has been turned off in GSL. Range checking is used primarily in the vector, matrix, and tensor `get()` and `set()` methods.

To see if range checking was turned on during installation (without calling the error handler), use `lib_settings_class::range_check()`.

Note that range checking in `O2scl` code is most often present in header files, rather than in source code. This means that range checking can be turned on or off in user-defined functions separately from whether or not it was used in the library classes and functions.

1.7.12 Shallow and deep copy

Copying `O2scl` vectors using constructors or the `=` operator 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 shallow 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.

For generic deep vector and matrix copying, you can use the template functions `vector_copy()`, `matrix_copy()`. These would allow you, for example, to copy an `ovector` to a `std::vector<double>` object. These functions do not do any memory allocation so that must be handled beforehand by the user.

1.7.13 Vector and matrix arithmetic

Several operators are available as member functions of the corresponding template:

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_view += matrix_view`
- `matrix_view -= matrix_view`
- `matrix_view += scalar`
- `matrix_view -= scalar`
- `matrix_view *= scalar`

Binary operators like addition, subtraction, and matrix multiplication are also defined for `ovector`, `uvector`, and related objects. The generic template for a binary operator, e.g.

```
template<class vec_t> vec_t &operator+(vec_t &v1, vec_t &v2);
```

is difficult because the compiler has no way of distinguishing vector and non-vector classes. At the moment, this is solved by creating a define macro for the binary operators. In addition to the predefined operators for native classes, the user may also define binary operators for other classes using the same macros. For example,

```
O2SCL_OP_VEC_VEC_ADD(o2scl::ovector, std::vector<double>,
std::vector<double>)
```

would provide an addition operator for [ovector](#) and vectors from the Standard Template Library. A full list of the operators which are defined by default and the associated macros are detailed in the documentation for [vec_arith.h](#).

The GSL BLAS routines can also be used directly with [ovector](#) and [omatrix](#) objects.

Note that some of 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.

1.7.14 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 discouraged among unrelated classes, but is permissible here because [ovector_tlate](#) is a descendant of `gsl_vector`. In particular, this will not generate "type-punning" warnings in later gcc versions. 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;
```

1.7.15 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.

1.7.16 `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 of `O2scl` if you find a GSL function with this behavior.

1.7.17 Vector and matrix views

Views are slightly different than in GSL in that they are now implemented as parent classes. Effectively, this means that vector views are just the same as normal vectors, except that they have no memory allocation functions (because the memory is presumably owned by a different object or scope). 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;
```

can be replaced by

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

1.7.18 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) {
            O2SCL_ERR("Unallocated vector not sent to afunction().",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) { av=a; return 0; }
};

```

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) {
        // do something with a
        return 0;
    }
};

```

1.7.19 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.

1.7.20 Matrix structure

The matrices from `omatrix_tlate` 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 unreferenced. 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. The matrices from `umatrix_tlate` have a trailing dimension which is always equal to the number of columns.

1.7.21 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.

1.7.22 Const-correctness with vectors

Vector objects, like `ovector`, can be const in the usual way. However because vector views, like `ovector_view` are like pointers to vectors, there are two ways in which they can be const. The view can point to a const vector, or the view can be a 'const view' which is fixed to point only to one vector (whether or not the vector pointed to happens to be const). This is exactly analogous to the distinction between `const double *p`, `double * const p`, and `const double * const p`. The first is a non-const pointer to const data, the second is a const pointer to non-const data, and the third is a const pointer to const data. The equivalent expressions in `O2scl` are

```
ovector_const_view v1;
const ovector_view v2;
const ovector_const_view v3;
```

Explicitly,

- `ovector_const_view` is a view of a const vector, the view may change, but the vector may not.
- `const ovector_const_view` is a const view of a const vector, the view may not point to a different vector and the vector may not change.
- `const ovector_view` is a const view of a normal vector, the view may not change, but the vector can. This same distinction is also present, for example, in `ublas` vectors views within `boost`.

A reference of type `ovector_base` is often used as a function argument, and can hold either a `ovector` or a `ovector_view`. The important rule to remember with `ovector_base` is that, a const reference, i.e.

```
const ovector_base &v;
```

is always a const reference to data which cannot be changed. A normal `ovector_base` reference does not refer to const data.

1.7.23 Vector and matrix output

Both the `ovector_view_tlate` and `omatrix_view_tlate` classes have an `<<` operator defined for each class. For writing generic vectors to a stream, you can use `vector_out()` which is defined in `vector.h`. Pretty matrix output is performed by global template functions `matrix_out()`, `matrix_cx_out_paren()`, and `matrix_out_paren()` which are defined in `columnify.h`.

1.7.24 Tensors

Some preliminary support is provided for tensors of arbitrary rank and size in the class `tensor`. Classes `tensor1`, `tensor2`, `tensor3`, and `tensor4` are rank-specific versions for 1-, 2-, 3- and 4-rank tensors. For n-dimensional data defined on a grid, `tensor_grid` provides a space to define a hyper-cubic grid in addition to the `tensor` data. This class `tensor_grid` also provides n-dimensional interpolation of the data defined on the specified grid.

1.8 Permutations

Permutations are implemented through the `permutation` class. This class is fully compatible with `gsl_permutation` objects since it is inherited from `gsl_permutation_struct`. The class also contains no new data members, so upcasting and downcasting can always be performed. It is perfectly permissible to call GSL `permutation` functions from `permutation` objects by simply passing the address of the `permutation`, i.e.

```
permutation p(4);
p.init();
gsl_permutation_swap(&p, 2, 3);
```

The functions which apply a `permutation` to a user-specified vector are member template functions in the `permutation` class (see `permutation::apply()`).

Memory allocation/deallocation between the class and the `gsl_struct` is compatible in many cases, but mixing these forms is strongly discouraged, i.e. avoid using `gsl_permutation_alloc()` on a `permutation` object, but rather use `permutation::allocate()` instead. The use of `permutation::free()` is encouraged, but any remaining memory will be deallocated in the object destructor.

1.9 Linear algebra

There is a small set of linear algebra routines. These are not intended to be a replacement for higher performance linear algebra libraries, but are designed for use by `O2scl` routines so that they work for generic matrix and vector types. For vector and matrix types using `operator[]`, the BLAS and linear algebra routines are inside the `o2scl_cblas` and `o2scl_linalg` namespaces. For vector and matrix types using `operator()`, the BLAS and linear algebra routines are inside the `o2scl_cblas_paren` and `o2scl_linalg_paren` namespaces.

The linear algebra classes and functions include:

- Householder transformations (`householder.h`)
- Householder solver (`hh.h`)
- LU decomposition and solver (`lu.h`)
- QR decomposition and solver (`qr.h`)
- Solve tridiagonal systems (`tridiag.h`)
- Lanczos diagonalization is inside class `lanczos`, which also can compute the eigenvalues of a tridiagonal matrix.
- Givens rotations (`givens.h`)

There is also a set of linear solvers for generic matrix and vector types which descend from `o2scl_linalg::linear_solver`. These classes provide GSL-like solvers, but are generalized so that they are compatible with vector and matrix types which allow access through `operator[]`.

For users who require high-performance linear algebra, the `ovector` and `omatrix` objects can be used to call LAPACK routines directly, just as can be done with GSL. For an example of how to do this, see <http://sourceware.org/ml/gsl-discuss/2001/msg00326.html>. For uBlas users, there is an example of how to use uBlas vectors with `O2scl` in [ublas vector example](#). Finally, there are also a couple of examples, `gesvd.cpp` and `zheev.cpp` in the `src/internal` directory which show how to call LAPACK with `O2scl` objects which may be adaptable for your platform and configuration.

1.10 Interpolation

The classes `o2scl_interp` and `o2scl_interp_vec` allow basic interpolation, lookup, differentiation, and integration of data given in two ovector or ovector views. In contrast to the GSL routines, data which is presented with a decreasing independent variable is handled automatically. For interpolation with arrays rather than ovector, use `array_interp` or `array_interp_vec`.

For fast interpolation of arrays where the independent variable is strictly increasing and without error-checking, you can directly use the children of `base_interp`.

1.10.1 The two interpolation interfaces

The difference between the two classes, `o2scl_interp` and `o2scl_interp_vec`, analogous to the difference between using `gsl_interp_eval()` and `gsl_spline_eval()` in GSL. You can create a `o2scl_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
o2scl_interp oi;
double y_half=oi.interp(0.5,20,x,y);
```

Alternatively, you can create a `o2scl_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
o2scl_interp_vec oi(20,x,y);
double y_half=oi.interp(0.5);
```

1.10.2 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.

1.10.3 "Smart" interpolation

The classes `smart_interp` and `smart_interp_vec` allow interpolation, lookup, differentiation, and integration of data which is non-monotonic or multiply-valued outside the region of interest. As with `o2scl_interp` above, the corresponding array versions are given in `sma_interp` and `sma_interp_vec`.

1.10.4 Lower-level interpolation objects

There are five lower-level interpolation objects which specify the basic `O2scl` interpolation types. They are

- [linear_interp](#)
- [cspline_interp](#)
- [cspline_peri_interp](#)
- [akima_interp](#)
- [akima_peri_interp](#)

1.10.5 Interpolation manager objects

Many `O2scl` classes require the ability to create and destroy interpolation objects at will, given one of the lower-level interpolation types described above. For this purpose, interpolation managers have been created, which allow the user to provide a class with an interpolation manager associated with a low-level interpolation type (and thus free the user from having to worry about the associated memory management). The abstract base interpolation manager type is [base_interp_mgr](#), and the default interpolation manager object is [def_interp_mgr](#). The [def_interp_mgr](#) class has two template arguments: the first is the vector type to be interpolated, and the second is the lower-level interpolation type.

For an example usage of the default interpolation manager, see the [Contour lines example](#), which specifies an interpolation manager for the [contour](#) class. The [table](#) class also works with interpolation manager objects (see [table::set_interp\(\)](#)).

1.10.6 Two and higher dimensional interpolation

Support for two-dimensional interpolation (by successive one-dimensional interpolations) is given in [twod_intp](#) (see [Two-dimensional Interpolation](#)), and a simple version of n-dimensional interpolation in [tensor_grid](#).

1.11 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_mkgsa](#), and [gsl_num](#). Some additional constants are given in the namespace [o2scl_const](#). Some of the numerical values have been updated from recently released data from NIST.

1.12 Function Objects

Functions are passed to numerical routines using template-based function classes, sometimes called "functors". 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
 - [jac_funct](#) : Jacobian function for solver
 - [grad_funct](#) : Gradient function for minimizers
 - [ool_hfunct](#) : Hessian product for constrained minimization
-

For each of these classes (except [funct](#)), there is a version named `_vfunct` instead of `_funct` which is designed to be used with C-style arrays instead of vector classes.

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
- `_cmfptr`: function pointer template for a class member function which is const
- `_strings`: functions specified using strings, e.g. "x^2-2", which are in the separate `O2scl_ext` package.
- `_noerr`: (for [funct](#) and [multi_funct](#)) a function which directly returns the function value rather than returning an integer error value
- `_npar`: (for [funct](#)) a function which has no parameters and directly returns the function value.

See the [Function and solver example](#) and the [Multidimensional solver example](#) which provide detailed examples of how functions can be specified to classes through these function objects.

There is a small overhead associated with the indirection: a "user class" accesses the function class which then calls function which was specified in the constructor of the function class. In many problems, the overhead associated with the indirection is small. Some of this overhead can always be avoided by inheriting directly from the function class and thus the user class will make a direct virtual function call. To eliminate the overhead entirely, one can specify a new type for the template parameter in the user class.

Note that virtual functions can be specified through this mechanism as well. For example, if [cern_mroot](#) is used to solve a set of equations specified as

```
class my_type_t {
    virtual member_func();
};
my_type_t my_instance;
class my_derived_type_t : public my_type_t {
    virtual member_func();
};
my_derived_type_t my_inst2;
mm_funct_mfptr<my_type_t> func(&my_inst2, &my_instance::member_func);
```

Then the solver will solve the member function in the derived type, not the parent type.

Note also that providing a user access to a function object instantiated with a protected or private member function is (basically) the same as providing them access to that function.

1.13 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.

1.14 String manipulation

There are a couple classes and functions to help manipulate strings of text. Conversion routines for `std::string` objects are given in [string_conv.h](#) and include

- `ptos()` - pointer to string

- [itos\(\)](#) - integer to string
- [dtos\(\)](#) - double to string
- [stoi\(\)](#) - string to integer
- [stod\(\)](#) - string to double

See also [size_of_exponent\(\)](#), [format_float](#), and [double_to_ieee_string\(\)](#).

A class called [columnify](#) converts a set of strings into nicely formatted columns by padding with the necessary amount of spaces. This class operates on string objects of type `std::string`, and also works well for formatting columns of floating-point numbers. This class is used to provide output for matrices in the functions [matrix_out\(\)](#), [matrix_out_paren\(\)](#), and [matrix_cx_out_paren\(\)](#). For output of vectors, see [vector_out\(\)](#) in [array.h](#).

A related function, [screenify\(\)](#), reformats a column of strings into many columns stored row-by-row in a new string array. It operates very similar to the way the classic Unix command `ls` organizes files and directories in multiple columns in order to save screen space.

The function [count_words\(\)](#) counts the number of "words" in a string, which are delimited by whitespace.

1.15 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.

1.16 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 ∞ : [gsl_inte_qagiu](#)
- General integration from $-\infty$ to 0: [gsl_inte_qagil](#)
- General integration from $-\infty$ to ∞ : [gsl_inte_qagi](#)
- 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_cos](#) and [gsl_inte_qawo_sin](#)
- Fourier integrals: [gsl_inte_qawf_cos](#) and [gsl_inte_qawf_sin](#)
- 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](#).

For the GSL-based integration routines, 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`. In particular, the integration classes attempt to ensure that

$$|\text{result} - I| \leq \text{Max}(\text{tolx}, \text{tolf}|I|)$$

and returns an error 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 the corresponding descendant of `inte::integ()` returns success, these inequalities may fail for sufficiently difficult functions. All of the GSL integration routines except for `gsl_inte_qng` use a workspace given in `gsl_inte_table` which holds the results of the various subdivisions of the original interval. The size of this workspace (and thus then number of subdivisions) can be controlled with `gsl_inte_table::set_workspace()`.

The GSL routines were originally based on QUADPACK, which is available at <http://www.netlib.org/quadpack>.

Note:

The GSL integration routines can sometimes lose precision if the integrand is everywhere much smaller than unity. Some rescaling is required in these cases.

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](#)).

1.17 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 end with the suffix `_complex` (`quadratic_complex`, `cubic_complex`, `quartic_complex`, and `poly_complex`). As a reminder, complex roots may not occur in conjugate pairs if the coefficients are not real. Most of these routines will return an error if the leading coefficient is zero.

In the public interfaces to the polynomial solvers, the complex type `std::complex<double>` is used. These can be converted to and from the GSL complex type using the `complex_to_gsl()` and `gsl_to_complex()` functions.

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, use `quadratic_std_complex`. 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_real2` tends to work 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 `simple_quartic_complex`.

1.18 Equation Solving

1.18.1 One-dimensional solvers

Solution of one equation in one variable is accomplished by children of the class `root`.

For one-dimensional solving, use `cern_root` or `gsl_root_brent` if you have the `root` bracketed, or `gsl_root_stef` if you have the derivative available. If you have neither a bracket or a derivative, you can use `cern_mroot_root`.

The `root` base class provides the structure for three different solving methods:

- `root::solve()` which solves a function given an initial guess x
- `root::solve_bkt()` which solves a function given a solution bracketed between x_1 and x_2 . The values of the function at x_1 and x_2 should have different signs.
- `root::solve_de()` 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 the `root` base class 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. Of course, it is frequently most efficient to use the solver in the way it was intended.

1.18.2 Multi-dimensional solvers

Solution of more than one equation is accomplished by descendants of the class `mroot`. The higher-level interface is provided by the function `mroot::msolve()`.

For multi-dimensional solving, you can use either `cern_mroot` or `gsl_mroot_hybrids`. While `cern_mroot` cannot utilize user-supplied derivatives, `gsl_mroot_hybrids` can use user-supplied derivative information (as in the GSL `hybridsj` method) using the function `gsl_mroot_hybrids::msolve_de()`.

1.19 Minimization

One-dimensional minimization is performed by descendants of `minimize`. There are two one-dimensional minimization algorithms, `cern_minimize` and `gsl_min_brent`, and they are both bracketing algorithms type where an interval and an initial guess must be provided. If only an initial guess and no bracket is given, these two classes will attempt to find a suitable bracket from the initial guess. While the `minimize` base class is designed to allow future descendants to optionally use derivative information, this is not yet supported for any one-dimensional minimizers.

Multi-dimensional minimization is performed by descendants of `multi_min`: `gsl_mmin_simp2`, `gsl_mmin_conp`, `gsl_mmin_conf`, and `gsl_mmin_bfgs2`. (The class `gsl_mmin_simp2` has been updated with the new "simplex2" method from GSL-1.12. The older "simplex" method is also available in `gsl_mmin_simp`.) The classes `gsl_mmin_simp` and `gsl_mmin_simp2` do not require any derivative information. The remaining minimization classes are intended for use when the `gradient` of the function is available, but they can also automatically compute the gradient numerically. The standard way to provide the gradient is to use a child of `grad_func` (or `grad_vfunc`). Finally, the user may specify the automatic `gradient` object of type `gradient` (or `gradient_array`) which is used by the minimizer to compute the `gradient` numerically when a function is not specified.

See an example for the usage of the multi-dimensional minimizers in [Multidimensional minimizer example](#).

Simulated annealing methods are also provided for multi-dimensional minimization (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 indeed found a true minimum.

The class `multi_min_fix` provides a convenient way of fixing some of the parameters and minimizing over others, without requiring a the function interface to be rewritten. An example is given in [Minimizer fixing variables example](#).

A naive way of implementing constraints is to add a function to the original which increases the value outside of the allowed region. This can be done with the functions `constraint()` and `lower_bound()`. There are two analogous functions, `cont_constraint()` and `cont_lower_bound()`, which continuous and differentiable versions. Where possible, it is better to use the constrained minimization routines described below.

1.20 Constrained Minimization

O₂scl reimplements the Open Optimization Library (OOL) available at <http://ool.sourceforge.net>. The associated classes allow constrained minimization when the constraint can be expressed as a hyper-cubic constraint on all of the independent variables. The routines have been rewritten and reformatted for C++ in order to facilitate the use of member functions and user-defined vector types as arguments. The base class is `ool_constr_mmin` and there are two different constrained minimization algorithms implemented in `ool_mmin_pgrad`, `ool_mmin_spg`. (The `ool_mmin_gencan` minimizer is not yet finished). The O₂scl implementation should be essentially identical to the most recently released version of OOL.

The constrained minimization classes operate in a similar way to the other multi-dimensional minimization classes (which are derived from `multi_min`). The constraints are specified with the function

```
ool_constr_mmin::set_constraints(size_t nc, vec_t &lower,
vec_t &upper);
```

and the minimization can be performed by calling either `multi_min::mmin()` or `multi_min::mmin_de()` (if the gradient is provided by the user). The method in `ool_mmin_gencan` requires a Hessian vector product and the user can specify this product for the minimization by using `ool_constr_mmin::mmin_hess()`. The Hessian product function can be specified as an object of type `ool_hfunct` or `ool_hvfunct` in a similar way to the other function objects in O₂scl.

There are five error codes defined in `ool_constr_mmin` which are specific to the OOL classes.

1.21 Monte Carlo Integration

Monte Carlo integration is performed by descendants of `mcarlo_inte` (`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.

1.22 Simulated Annealing

Minimization by simulated annealing is performed by descendants of `sim_anneal` (see `gsl_anneal`). Because simulated annealing is particularly well-suited to parallelization, a multi-threaded minimizer analogous to `gsl_anneal` is given in `anneal_mt`. This header-only class uses the Boost libraries and requires it for use.

1.23 Non-linear Least-Squares Fitting

Fitting is performed by descendants of `fit_base` and fitting functions can be specified using `fit_funct`. 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.

1.24 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`. To specify a set of functions to these classes, use a child of `ode_funct` for a generic vector type or a child of `ode_vfunct`

when using arrays.

Solution of simple initial value problems is performed by [ode_iv_solve](#). This class contains a couple different methods, depending on whether the user needs only the final value or the solution on a fixed grid.

The solution of boundary-value problems is based on the abstract base class [ode_bv_solve](#). At the moment, a simple shooting method is the only implementation of this base class and is given in [ode_bv_shoot](#). An experimental multishooting class is given in [ode_bv_multishoot](#).

An experimental application of linear solvers to solve the finite-difference equations for a boundary value problem is given in [ode_it_solve](#).

1.25 Random Number Generation

Random number generators are descendants of [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_rnga](#) to avoid an additional performance penalty. This may not be a truly "object-oriented" interface in that it does not use virtual functions, but it avoids any possible performance penalty. Random number generators are implemented as templates in [sim_anneal](#) and [mcarlo_inte](#). In these classes, the random number generator is a template type, rather than a member data pointer, in order to ensure fast execution.

1.26 Two-dimensional Interpolation

Successive use of [smart_interp](#) is implemented in [twod_intp](#), which is useful for interpolating data presented on a two-dimensional grid (though the spacings between grid points need not be equal). Also, one can compute contour lines from data represented on a grid using the [contour](#) class.

If data is arranged without a grid, then [planar_intp](#) can be used. At present, the only way to compute contour lines on data which is not defined on a grid is to use [planar_intp](#) to recast the data on a grid and then use [contour](#) afterwards.

Higher-dimensional interpolation is possible with [tensor_grid](#).

1.27 Chebyshev Approximation

A class implementing the Chebyshev approximations based on GSL is given in [gsl_chebapp](#). This class has its own copy constructor, so that Chebyshev approximations can be copied and passed as arguments to functions.

1.28 Unit Conversions

There is a class which performs conversion between units specified with strings in [convert_units](#). The [convert_units](#) class also uses a system call to the GNU units command (if it's installed in the path) to obtain the proper conversion factors.

1.29 Other classes and functions

The O₂scl library contains several classes which are still under development and are to be considered experimental. While it is expected that the testing code associated with these classes will succeed on most any system, the interface, structure, and basic approaches used by these classes may change in future versions of O₂scl.

Three-dimensional data tables - [table3d](#)

Two-dimensional Gaussian distribution - [gaussian_2d](#)

Series acceleration - [gsl_series](#)

Command-line interface - [cli](#)

Automatic bin sizing - [bin_size](#)

Fourier transforms - [gsl_fft](#)

Polylogarithms - [polylog](#)

1.30 Library settings

There are a couple library settings which are handled by a global object [lib_settings](#) of type [lib_settings_class](#).

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 after installation, 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 data directory has not changed.

1.31 Object I/O

The I/O portion of the library is still 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);
```

1.31.1 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, [cinput::object_in_name\(\)](#), [cinput::object_in_mem\(\)](#), and [cinput::object_in_mem_name\(\)](#) allow the freedom to input an object with a name or with memory allocation. The function [coutput::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.

1.32 Example source code

1.32.1 Example list

- [Function and solver example](#) shows how member functions and external parameters are supplied to O₂scl numerical routines. In this case, a member function representing an equation with one unknown is solved using a one-dimensional solver.
- [Multidimensional solver example](#) demonstrates the multidimensional function solver and several different methods of specifying the function to be solved.

- [Multidimensional minimizer example](#)
- [Minimizer fixing variables example](#)
- [Numerical differentiation example](#)
- [Numerical integration example](#)
- [Ordinary differential equations example](#)
- [Simulated annealing example](#) demonstrates multidimensional minimization by simulated annealing
- [Multidimensional integration example](#) demonstrates several ways to compute a multidimensional integral including Monte Carlo and direct methods
- [Contour lines example](#)
- [Two-dimensional interpolation example](#)
- [ublas vector example](#) demonstrates the use of interpolation and solver classes with ublas vectors

1.32.2 Function and solver example

```

/* Example: ex_fptr.cpp
-----
This gives an example of the how member functions and external
parameters are supplied to numerical routines. In this case, a
member function with two parameters is passed to the gsl_root_brent
class, which solves the equation. One of the parameters is member
data, and the other is specified using the extra parameter argument
to the function.
*/

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

using namespace std;
using namespace o2scl;

class my_class {
private:
    double parameter;

public:
    void set_parameter() { parameter=0.01; }

    // A function demonstrating the different ways of implementing
    // function parameters
    double function_to_solve(double x, double &p) {
        return atan((x-parameter)*4)*(1.0+sin((x-parameter)*50.0)/p);
    }
};

// A simple function to make the plot
int plot(double sol);

int main(void) {

    cout.setf(ios::scientific);

    test_mgr t;
    // Only print something out if one of the tests fails

```

```

t.set_output_level(1);

// 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=-1;
double x2=2;
double p=1.1;

// 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;

// Write the function being solved to a file (see source code
// in examples directory for details)
plot(x1);

t.report();
return 0;
}
// End of example

```

The image below shows how the solver progresses to the solution of the example function.

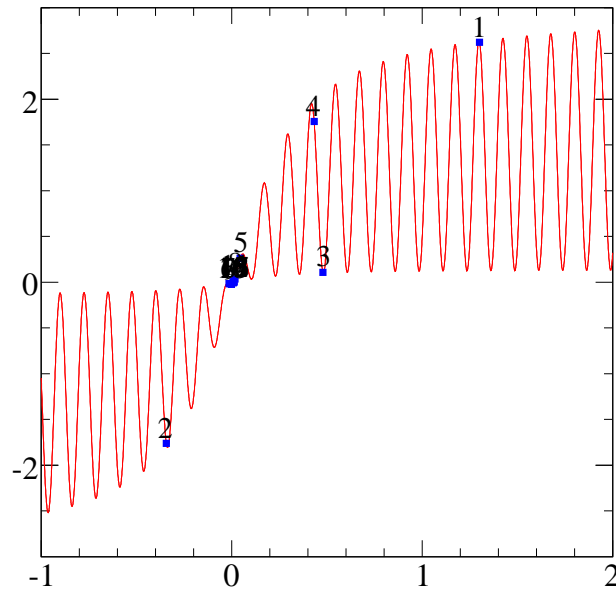


Figure 1: ex_fptr.eps

1.32.3 Multidimensional solver example

```

/* Example: ex_mroot.cpp
-----
Several ways to use an O2scl solver to solve a simple function
*/

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/mm_funct.h>
#include <o2scl/gsl_mroot_hybrids.h>
#include <o2scl/cern_mroot.h>

using namespace std;
using namespace o2scl;

int gfn(size_t nv, const ovector_base &x,
        ovector_base &y, void *&pa) {
    y[0]=sin(x[1]-0.2);
    y[1]=sin(x[0]-0.25);
    return 0;
}

class cl {

public:

    // Store the number of function and derivative evaluations
    int nf, nd;

    int mfn(size_t nv, const ovector_base &x, ovector_base &y,
            void *&pa) {
        y[0]=sin(x[1]-0.2);
        y[1]=sin(x[0]-0.25);
        nf++;
        return 0;
    }
}

```

```

int operator()(size_t nv, const ovector_base &x,
               ovector_base &y, void *&pa) {
    y[0]=sin(x[1]-0.2);
    y[1]=sin(x[0]-0.25);
    nf++;
    return 0;
}

int mfnd(size_t nv, ovector_base &x, ovector_base &y,
         omatrix_base &j, void *&pa) {
    j[0][0]=0.0;
    j[0][1]=cos(x[1]-0.2);
    j[1][0]=cos(x[0]-0.25);
    j[1][1]=0.0;
    nd++;
    return 0;
}

int mfna(size_t nv, const double x[2], double y[2], void *&pa) {
    y[0]=sin(x[1]-0.2);
    y[1]=sin(x[0]-0.25);
    return 0;
}

int mfnad(size_t nv, double x[], double y[], double j[2][2], void *&pa) {
    j[0][0]=0.0;
    j[0][1]=cos(x[1]-0.2);
    j[1][0]=cos(x[0]-0.25);
    j[1][1]=0.0;
    return 0;
}

};

int main(void) {
    cl acl;
    ovector x(2);
    double xa[2];
    int i;
    void *vp=NULL;
    size_t tmp;
    int r1, r2, r3;
    bool done;
    test_mgr t;

    t.set_output_level(1);

    /*
     * Using a member function with \ref ovector objects
     */
    mm_funct_mfpnr<cl,void *,ovector_base> f1(&acl,&cl::mfnd);
    gsl_mroot_hybrids<void *> cr1;

    x[0]=0.5;
    x[1]=0.5;
    acl.nf=0;
    int ret1=cr1.msolve(2,x,vp,f1);
    cout << "GSL solver (numerical Jacobian): " << endl;
    cout << "Return value: " << ret1 << endl;
    cout << "Number of iterations: " << cr1.last_ntrial << endl;
    cout << "Number of function evaluations: " << acl.nf << endl;
    cout << endl;
    t.test_rel(x[0],0.25,1.0e-6,"1a");
    t.test_rel(x[1],0.2,1.0e-6,"1b");

    /*
     * Using the CERNLIB solver
     */
    cern_mroot<void *> cr2;

```

```

x[0]=0.5;
x[1]=0.5;
acl.nf=0;
int ret2=cr2.msolve(2,x,vp,f1);
cout << "CERNLIB solver (numerical Jacobian): " << endl;
cout << "Return value: " << ret2 << endl;
cout << "INFO parameter: " << cr2.get_info() << endl;
cout << "Number of function evaluations: " << acl.nf << endl;
cout << endl;
t.test_rel(x[0],0.25,1.0e-6,"2a");
t.test_rel(x[1],0.2,1.0e-6,"2b");

/*
  Using a member function with \ref ovector objects, but
  using the GSL-like interface with set() and iterate().
*/
gsl_mroot_hybrids<void *> cr3;

x[0]=0.5;
x[1]=0.5;
cr3.allocate(2);
cr3.set(2,x,f1,vp);
done=false;
do {
  r3=cr3.iterate();
  double resid=fabs(cr3.f[0])+fabs(cr3.f[1]);
  if (resid<cr3.tolf || r3>0) done=true;
} while (done==false);
t.test_rel(cr3.x[0],0.25,1.0e-6,"3a");
t.test_rel(cr3.x[1],0.2,1.0e-6,"3b");
cr3.free();

/*
  Now instead of using the automatic Jacobian, using
  a user-specified Jacobian.
*/
jac_funct_mfp<cl,void *,ovector_base,omatrix_base> j4(&acl,&cl::mfnd);

x[0]=0.5;
x[1]=0.5;
acl.nf=0;
acl.nd=0;
int ret4=cr1.msolve_de(2,x,vp,f1,j4);
cout << "GSL solver (analytic Jacobian): " << endl;
cout << "Return value: " << ret4 << endl;
cout << "Number of iterations: " << cr1.last_ntrial << endl;
cout << "Number of function evaluations: " << acl.nf << endl;
cout << "Number of Jacobian evaluations: " << acl.nd << endl;
cout << endl;
t.test_rel(x[0],0.25,1.0e-6,"4a");
t.test_rel(x[1],0.2,1.0e-6,"4b");

/*
  Using a user-specified Jacobian and the GSL-like interface
*/
gsl_mroot_hybrids<void *> cr5;

x[0]=0.5;
x[1]=0.5;
cr5.allocate(2);
cr5.set_de(2,x,f1,j4,vp);
done=false;
do {
  r3=cr5.iterate();
  double resid=fabs(cr5.f[0])+fabs(cr5.f[1]);
  if (resid<cr5.tolf || r3>0) done=true;
} while (done==false);
t.test_rel(cr5.x[0],0.25,1.0e-6,"5a");
t.test_rel(cr5.x[1],0.2,1.0e-6,"5b");
cr5.free();

```



```

/*
    Using C-style arrays instead of ovector objects
*/
mm_vfunct_mfp_ptr<cl,void *,2> f6(&acl,&cl::mfna);
gsl_mroot_hybrids<void *,mm_vfunct_mfp_ptr<cl,void *,2>,double[2],
    double[2],array_alloc<double[2]> > cr6;

xa[0]=0.5;
xa[1]=0.5;
cr6.msolve(2,xa,vp,f6);
t.test_rel(xa[0],0.25,1.0e-6,"6a");
t.test_rel(xa[1],0.2,1.0e-6,"6b");

/*
    Using the CERNLIB solver with C-style arrays instead of ovector objects
*/
cern_mroot<void *,mm_vfunct_mfp_ptr<cl,void *,2>,double[2],
    double[2],array_alloc<double[2]> > cr7;

xa[0]=0.5;
xa[1]=0.5;
cr7.msolve(2,xa,vp,f6);
t.test_rel(xa[0],0.25,1.0e-6,"7a");
t.test_rel(xa[1],0.2,1.0e-6,"7b");

/*
    Using C-style arrays with a user-specified Jacobian
*/
jac_vfunct_mfp_ptr<cl,void *,2> j8(&acl,&cl::mfna_d);
gsl_mroot_hybrids<void *,mm_vfunct_mfp_ptr<cl,void *,2>,double[2],
    double[2],array_alloc<double[2]>,double[2][2],double[2][2],
    array_2d_alloc<double[2][2]>,jac_vfunct<void *,2> > cr8;

xa[0]=0.5;
xa[1]=0.5;
cr8.msolve_de(2,xa,vp,f6,j8);
t.test_rel(xa[0],0.25,1.0e-6,"8a");
t.test_rel(xa[1],0.2,1.0e-6,"8b");

/*
    Using a class with an operator(). Note that there can be only one
    operator() function in each class.
*/
gsl_mroot_hybrids<void *,cl,ovector_base> cr9;

x[0]=0.5;
x[1]=0.5;
cr9.msolve(2,x,vp,acl);
t.test_rel(x[0],0.25,1.0e-6,"9a");
t.test_rel(x[1],0.2,1.0e-6,"9b");

/*
    Using a function pointer to a global function.
*/
typedef int (*gfnt)(size_t, const ovector_base &, ovector_base &,
    void *&);
gsl_mroot_hybrids<void *,gfnt,ovector_base> cr10;
gfnt f10=&gfn;

x[0]=0.5;
x[1]=0.5;
cr10.msolve(2,x,vp,f10);
t.test_rel(x[0],0.25,1.0e-6,"10a");
t.test_rel(x[1],0.2,1.0e-6,"10b");

t.report();
return 0;
}
// End of example

```

1.32.4 Multidimensional minimizer example

```

/* Example: ex_mmin.cpp
-----
Example usage of the multidimensional minimizers
*/

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/multi_funct.h>
#include <o2scl/gsl_mmin_simp2.h>
#include <o2scl/gsl_mmin_conf.h>
#include <o2scl/gsl_mmin_conp.h>
#include <o2scl/gsl_mmin_bfgs2.h>

using namespace std;
using namespace o2scl;

class cl {

public:

    int mfn(size_t nv, const ovector_base &x, double &y, int &pa) {
        y=(x[0]-2.0)*(x[0]-2.0)+(x[1]-1.0)*(x[1]-1.0);
        return 0;
    }

};

int main(void) {
    cl acl;
    ovector x(2);
    int vp=0;
    double fmin;
    test_mgr t;

    t.set_output_level(1);
    cout.setf(ios::scientific);

    /*
    Using a member function with \ref ovector objects
    */
    multi_funct_mfptr<cl,int,ovector_base> f1(&acl,&cl::mfn);
    gsl_mmin_simp2<int> gm1;
    gsl_mmin_conf<int> gm2;
    gsl_mmin_conp<int> gm3;
    gsl_mmin_bfgs2<int> gm4;

    x[0]=0.5;
    x[1]=0.5;
    gm1.mmin(2,x,fmin,vp,f1);
    cout << gm1.last_ntrial << endl;
    cout << "Found minimum at: " << x << endl;
    t.test_rel(x[0],2.0,1.0e-4,"1a");
    t.test_rel(x[1],1.0,1.0e-4,"1b");

    x[0]=0.5;
    x[1]=0.5;
    gm2.mmin(2,x,fmin,vp,f1);
    cout << gm2.last_ntrial << endl;
    cout << "Found minimum at: " << x << endl;
    t.test_rel(x[0],2.0,1.0e-4,"2a");
    t.test_rel(x[1],1.0,1.0e-4,"2b");

    x[0]=0.5;
    x[1]=0.5;
    gm3.mmin(2,x,fmin,vp,f1);
    cout << gm3.last_ntrial << endl;
    cout << "Found minimum at: " << x << endl;
    t.test_rel(x[0],2.0,1.0e-4,"3a");

```

```

t.test_rel(x[1],1.0,1.0e-4,"3b");

x[0]=0.5;
x[1]=0.5;
gm4.err_nonconv=false;
gm4.mmin(2,x,fmin,vp,f1);
cout << gm4.last_ntrial << endl;
cout << "Found minimum at: " << x << endl;
t.test_rel(x[0],2.0,1.0e-4,"4a");
t.test_rel(x[1],1.0,1.0e-4,"4b");

t.report();
return 0;
}
// End of example

```

1.32.5 Minimizer fixing variables example

```

/* Example: ex_mmin_fix.cpp
-----
Example usage of the mmin_fix class, which fixes some of the
paramters for a multidimensional minimization.
*/

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/multi_funct.h>
#include <o2scl/gsl_mmin_simp2.h>
#include <o2scl/multi_min_fix.h>

using namespace std;
using namespace o2scl;

class cl {

public:

    int mfn(size_t nv, const ovector_base &x, double &y, int &pa) {
        y=(x[0]-2.0)*(x[0]-2.0)+(x[1]-1.0)*(x[1]-1.0)+x[2]*x[2];
        return 0;
    }

};

int main(void) {
    cl acl;
    ovector x(3);
    int vp=0;
    double fmin;
    test_mgr t;

    t.set_output_level(1);
    cout.setf(ios::scientific);

    /*
       Perform the minimization the standard way, with the
       simplex2 minimizer
    */
    multi_funct_mfptr<cl,int,ovector_base> f1(&acl,&cl::mfn);
    gsl_mmin_simp2<int> gm1;

    x[0]=0.5;
    x[1]=0.5;
    x[2]=0.5;
    gm1.mmin(3,x,fmin,vp,f1);
    cout << gm1.last_ntrial << " iterations." << endl;
    cout << "Found minimum at: " << x << endl;
    t.test_rel(x[0],2.0,1.0e-4,"1a");

```

```

t.test_rel(x[1],1.0,1.0e-4,"1b");
t.test_rel(x[2],0.0,1.0e-4,"1c");

// Create a new multi_min_fix object
multi_min_fix<int,bool[3]> gmf;

// Create a base minimizer which can be used by the multi_min_fix
// object. Note that we can't use 'gml' here, because it has a
// different type than 'gm2', even though its functionality is
// effectively the same.
gsl_mmin_simp2<int,multi_funcnt_mfptr
    <multi_min_fix<int,bool[3]>,int> > gm2;

// Set the base minimizer
gmf.set_mmin(gm2);

/*
    First perform the minimization as above.
*/
x[0]=0.5;
x[1]=0.5;
x[2]=0.5;
gmf.mmin(3,x,fmin,vp,f1);
cout << gmf.last_ntrial << " iterations." << endl;
cout << "Found minimum at: " << x << endl;
t.test_rel(x[0],2.0,1.0e-4,"2a");
t.test_rel(x[1],1.0,1.0e-4,"2b");
t.test_rel(x[2],0.0,1.0e-4,"2c");

/*
    Now fix the 2nd variable, and re-minimize.
*/
bool fix[3]={false,true,false};
x[0]=0.5;
x[1]=0.5;
x[2]=0.5;
gmf.mmin_fix(3,x,fmin,fix,vp,f1);
cout << gmf.last_ntrial << " iterations." << endl;
cout << "Found minimum at: " << x << endl;
t.test_rel(x[0],2.0,1.0e-4,"3a");
t.test_rel(x[1],0.5,1.0e-4,"3b");
t.test_rel(x[2],0.0,1.0e-4,"3c");

t.report();
return 0;
}
// End of example

```

1.32.6 Numerical differentiation example

```

/* Example: ex_deriv.cpp
-----
    An example to demonstrate numerical differentiation
*/

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/funcnt.h>
#include <o2scl/gsl_deriv.h>
#include <o2scl/cern_deriv.h>

using namespace std;
using namespace o2scl;

class cl {
public:

```

```

// This is the function we'll take the derivative of
double function(double x) {
    return sin(2.0*x)+0.5;
}
};

int main(void) {
    cl acl;
    ovector x(2);
    double xa[2];
    int i;
    void *vp=0;
    size_t tmp;
    int r1, r2, r3;
    bool done;

    test_mgr t;
    t.set_output_level(2);

    funct_mfptr_nopar<cl,void *> f1(&acl,&cl::function);

    gsl_deriv<void *> gd;
    // Note that the GSL derivative routine requires an initial stepsize
    gd.h=1.0e-3;
    cern_deriv<void *> cd;

    // Compute the first derivative using the gsl_deriv class and
    // verify that the answer is correct
    double d1=gd.calc(1.0,vp,f1);
    t.test_rel(d1,2.0*cos(2.0),1.0e-10,"gsl_deriv");

    // Compute the first derivative using the cern_deriv class and
    // verify that the answer is correct
    double d2=cd.calc(1.0,vp,f1);
    t.test_rel(d2,2.0*cos(2.0),1.0e-10,"cern_deriv");

    // Compute the second derivative also
    double d3=gd.calc2(1.0,vp,f1);
    t.test_rel(d3,-4.0*sin(2.0),5.0e-7,"gsl_deriv");

    double d4=cd.calc2(1.0,vp,f1);
    t.test_rel(d4,-4.0*sin(2.0),1.0e-8,"cern_deriv");

    t.report();
    return 0;
}
// End of example

```

1.32.7 Numerical integration example

```

/* Example: ex_inte.cpp
-----
An example to demonstrate numerical integration.
*/

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/constants.h>
#include <o2scl/funct.h>
#include <o2scl/gsl_inte_qag.h>
#include <o2scl/gsl_inte_qagi.h>
#include <o2scl/gsl_inte_qagiu.h>
#include <o2scl/gsl_inte_qagil.h>
#include <o2scl/cern_adapt.h>

using namespace std;
using namespace o2scl;
using namespace o2scl_const;

```

```

class cl {
public:

    // We'll use this to count the number of function
    // evaluations required by the integration routines
    int nf;

    // A function to be integrated
    double integrand(double x) {
        nf++;
        return exp(-x*x);
    }

    // Another function to be integrated
    double integrand2(double x) {
        nf++;
        return sin(2.0*x)+0.5;
    }
};

int main(void) {
    cl acl;
    void *vp=0;
    test_mgr t;

    t.set_output_level(1);

    funct_mfptr_nopar<cl,void *> f1(&acl,&acl::integrand);
    funct_mfptr_nopar<cl,void *> f2(&acl,&acl::integrand2);

    // We don't need to specify the function type in the integration
    // objects, because we're using the default function type (type
    // funct).
    gsl_inte_qag<void *> g;
    gsl_inte_qagi<void *> gi;
    gsl_inte_qagiu<void *> gu;
    gsl_inte_qagil<void *> gl;
    cern_adapt<void *> ca;

    // The result and the uncertainty
    double res, err;

    // An integral from -infinity to +infinity (the limits are ignored)
    acl.nf=0;
    int ret1=gi.integ_err(f1,0.0,0.0,vp,res,err);
    cout << "gsl_inte_qagi: " << endl;
    cout << "Return value: " << ret1 << endl;
    cout << "Result: " << res << " Uncertainty: " << err << endl;
    cout << "Number of iterations: " << gi.last_iter << endl;
    cout << "Number of function evaluations: " << acl.nf << endl;
    cout << endl;
    t.test_rel(res,sqrt(pi),1.0e-8,"inte 1");

    // An integral from 0 to +infinity (the second limit argument is
    // ignored in the line below)
    acl.nf=0;
    gu.integ_err(f1,0.0,0.0,vp,res,err);
    cout << "gsl_inte_qagiu: " << endl;
    cout << "Return value: " << ret1 << endl;
    cout << "Result: " << res << " Uncertainty: " << err << endl;
    cout << "Number of iterations: " << gu.last_iter << endl;
    cout << "Number of function evaluations: " << acl.nf << endl;
    cout << endl;
    t.test_rel(res,sqrt(pi)/2.0,1.0e-8,"inte 2");

    // An integral from -infinity to zero (the first limit argument is
    // ignored in the line below)
    acl.nf=0;

```

```

gl.integ_err(f1,0.0,0.0,vp,res,err);
cout << "gsl_inte_qagil: " << endl;
cout << "Return value: " << ret1 << endl;
cout << "Result: " << res << " Uncertainty: " << err << endl;
cout << "Number of iterations: " << gl.last_iter << endl;
cout << "Number of function evaluations: " << acl.nf << endl;
cout << endl;
t.test_rel(res,sqrt(pi)/2.0,1.0e-8,"inte 3");

// An integral from 0 to 1
acl.nf=0;
g.integ_err(f2,0.0,1.0,vp,res,err);
cout << "gsl_inte_qag: " << endl;
cout << "Return value: " << ret1 << endl;
cout << "Result: " << res << " Uncertainty: " << err << endl;
cout << "Number of iterations: " << g.last_iter << endl;
cout << "Number of function evaluations: " << acl.nf << endl;
cout << endl;
t.test_rel(res,0.5+sin(1.0)*sin(1.0),1.0e-8,"inte 4");

// An integral from 0 to 1
acl.nf=0;
ca.integ_err(f2,0.0,1.0,vp,res,err);
cout << "cern_adapt: " << endl;
cout << "Return value: " << ret1 << endl;
cout << "Result: " << res << " Uncertainty: " << err << endl;
cout << "Number of iterations: " << ca.last_iter << endl;
cout << "Number of function evaluations: " << acl.nf << endl;
cout << endl;
t.test_rel(res,0.5+sin(1.0)*sin(1.0),1.0e-8,"inte 5");

t.report();
return 0;
}
// End of example

```

1.32.8 Ordinary differential equations example

```

/* Example: ex_ode.cpp
-----
An example to demonstrate solving differential equations
*/

#include <gsl/gsl_sf_bessel.h>
#include <gsl/gsl_sfairy.h>
#include <gsl/gsl_sf_gamma.h>
#include <o2scl/test_mgr.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/ode_funct.h>
#include <o2scl/gsl_rkck.h>
#include <o2scl/gsl_rk8pd.h>
#include <o2scl/gsl_astep.h>
#include <o2scl/ode_iv_solve.h>

using namespace std;
using namespace o2scl;

// Differential equation defining the Bessel function. This assumes
// the second derivative at x=0 is 0 and thus only works for odd alpha.
int derivs(double x, size_t nv, const ovector_base &y,
           ovector_base &dydx, double &alpha) {
    dydx[0]=y[1];
    if (x==0.0) dydx[1]=0.0;
    else dydx[1]=(-x*y[1]+(-x*x+alpha*alpha)*y[0])/x/x;
    return 0;
}

// Differential equation defining the Airy Ai(x) function.

```

```

int derivs2(double x, size_t nv, const ovector_base &y,
            ovector_base &dydx, double &alpha) {
    dydx[0]=y[1];
    dydx[1]=y[0]*x;
    return 0;
}

int main(void) {
    int i;
    double x, dx=1.0e-1;
    ovector y(2), dydx(2), yout(2), yerr(2), dydx_out(2);
    void *vp=NULL;
    test_mgr t;
    t.set_output_level(1);

    cout.setf(ios::scientific);
    cout.setf(ios::showpos);

    ode_funct_fptr<double,ovector_base> od(derivs);
    ode_funct_fptr<double,ovector_base> od2(derivs2);
    gsl_rkck<double> ode;
    gsl_rk8pd<double> ode2;
    double alpha=1.0;

    // Solve using the non-adaptive Cash-Karp stepper.

    cout << "Bessel function, Cash-Karp: " << endl;
    x=0.0;
    y[0]=0.0;
    y[1]=0.5;
    derivs(x,2,y,dydx,alpha);
    cout << " x          J1(calc)          J1(exact)          rel. diff.          "
         << "err" << endl;
    while (x<1.0) {
        ode.step(x,dx,2,y,dydx,y,yerr,dydx,alpha,od);
        x+=dx;
        cout << x << " " << y[0] << " "
             << gsl_sf_bessel_J1(x) << " ";
        cout << fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)) << " ";
        cout << yerr[0] << endl;
        t.test_rel(y[0],gsl_sf_bessel_J1(x),5.0e-5,"rkck");
    }
    cout << "Accuracy at end: "
         << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
    cout << endl;

    // Solve using the non-adaptive Prince-Dormand stepper. Note that
    // for the Bessel function, the 8th order stepper performs worse
    // than the 4th order. The error returned by the stepper is
    // larger near x=0, as expected.

    cout << "Bessel function, Prince-Dormand: " << endl;
    x=0.0;
    y[0]=0.0;
    y[1]=0.5;
    derivs(x,2,y,dydx,alpha);
    cout << " x          J1(calc)          J1(exact)          rel. diff.          "
         << "err" << endl;
    while (x<1.0) {
        ode2.step(x,dx,2,y,dydx,y,yerr,dydx,alpha,od);
        x+=dx;
        cout << x << " " << y[0] << " "
             << gsl_sf_bessel_J1(x) << " ";
        cout << fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)) << " ";
        cout << yerr[0] << endl;
        t.test_rel(y[0],gsl_sf_bessel_J1(x),5.0e-4,"rk8pd");
    }
    cout << "Accuracy at end: "
         << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
    cout << endl;
}

```



```

// Solve using the non-adaptive Cash-Karp stepper.

cout << "Airy function, Cash-Karp: " << endl;
x=0.0;
y[0]=1.0/pow(3.0,2.0/3.0)/gsl_sf_gamma(2.0/3.0);
y[1]=-1.0/pow(3.0,1.0/3.0)/gsl_sf_gamma(1.0/3.0);
derivs2(x,2,y,dydx,alpha);
cout << " x          Ai(calc)          Ai(exact)          rel. diff.      "
    << "err" << endl;
while (x<1.0) {
    ode.step(x,dx,2,y,dydx,y,yerr,dydx,alpha,od2);
    x+=dx;
    cout << x << " " << y[0] << " "
        << gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE) << " ";
    cout << fabs((y[0]-gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE))/
        gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE)) << " ";
    cout << yerr[0] << endl;
    t.test_rel(y[0],gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE),1.0e-8,"rkck");
}
cout << "Accuracy at end: "
    << fabs(y[0]-gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE))/
    gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE) << endl;
cout << endl;

// Solve using the non-adaptive Prince-Dormand stepper. On this
// function, the higher-order routine performs significantly
// better.

cout << "Airy function, Prince-Dormand: " << endl;
x=0.0;
y[0]=1.0/pow(3.0,2.0/3.0)/gsl_sf_gamma(2.0/3.0);
y[1]=-1.0/pow(3.0,1.0/3.0)/gsl_sf_gamma(1.0/3.0);
derivs2(x,2,y,dydx,alpha);
cout << " x          Ai(calc)          Ai(exact)          rel. diff.      "
    << "err" << endl;
while (x<1.0) {
    ode2.step(x,dx,2,y,dydx,y,yerr,dydx,alpha,od2);
    x+=dx;
    cout << x << " " << y[0] << " "
        << gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE) << " ";
    cout << fabs((y[0]-gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE))/
        gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE)) << " ";
    cout << yerr[0] << endl;
    t.test_rel(y[0],gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE),1.0e-14,"rk8pd");
}
cout << "Accuracy at end: "
    << fabs(y[0]-gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE))/
    gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE) << endl;
cout << endl;

// Solve using the GSL adaptive stepper

cout << "Adaptive stepper: " << endl;
gsl_astep<double> ode3;
x=0.0;
y[0]=0.0;
y[1]=0.5;
cout << " x          J1(calc)          J1(exact)          rel. diff.;"
    << "err_0          err_1" << endl;
int k=0;
while (x<10.0) {
    int retx=ode3.astep(x,dx,10.0,2,y,dydx,yerr,alpha,od);
    if (k%3==0) {
        cout << retx << " " << x << " " << y[0] << " "
            << gsl_sf_bessel_J1(x) << " ";
        cout << fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)) << " ";
        cout << yerr[0] << " " << yerr[1] << endl;
    }
    t.test_rel(y[0],gsl_sf_bessel_J1(x),5.0e-3,"astep");
}

```

```

    t.test_rel(y[1], 0.5*(gsl_sf_bessel_J0(x)-gsl_sf_bessel_Jn(2,x)),
               5.0e-3, "astep 2");
    t.test_rel(yerr[0], 0.0, 4.0e-6, "astep 3");
    t.test_rel(yerr[1], 0.0, 4.0e-6, "astep 4");
    t.test_gen(retx==0, "astep 5");
    k++;
}
cout << "Accuracy at end: "
      << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

// Decrease the tolerances, and the adaptive stepper takes
// smaller step sizes.

cout << "Adaptive stepper with smaller tolerances: " << endl;
ode3.con.eps_abs=1.0e-8;
ode3.con.a_dydt=1.0;
x=0.0;
y[0]=0.0;
y[1]=0.5;
cout << "    x                J1(calc)        J1(exact)        rel. diff.";
cout << "    err_0            err_1" << endl;
k=0;
while (x<10.0) {
    int retx=ode3.astep(x,dx,10.0,2,y,dydx,yerr,alpha,od);
    if (k%3==0) {
        cout << retx << " " << x << " " << y[0] << " "
              << gsl_sf_bessel_J1(x) << " ";
        cout << fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)) << " ";
        cout << yerr[0] << " " << yerr[1] << endl;
    }
    t.test_rel(y[0],gsl_sf_bessel_J1(x),5.0e-3,"astep");
    t.test_rel(y[1],0.5*(gsl_sf_bessel_J0(x)-gsl_sf_bessel_Jn(2,x)),
               5.0e-3,"astep 2");
    t.test_rel(yerr[0],0.0,4.0e-8,"astep 3");
    t.test_rel(yerr[1],0.0,4.0e-8,"astep 4");
    t.test_gen(retx==0,"astep 5");
    k++;
}
cout << "Accuracy at end: "
      << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

// Use the higher-order stepper, and less steps are required. The
// stepper automatically takes more steps near x=0 in order since
// the higher-order routine has more trouble there.

cout << "Adaptive stepper, Prince-Dormand: " << endl;
ode3.set_step(ode2);
x=0.0;
y[0]=0.0;
y[1]=0.5;
cout << "    x                J1(calc)        J1(exact)        rel. diff.";
cout << "    err_0            err_1" << endl;
k=0;
while (x<10.0) {
    int retx=ode3.astep(x,dx,10.0,2,y,dydx,yerr,alpha,od);
    if (k%3==0) {
        cout << retx << " " << x << " " << y[0] << " "
              << gsl_sf_bessel_J1(x) << " ";
        cout << fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)) << " ";
        cout << yerr[0] << " " << yerr[1] << endl;
    }
    t.test_rel(y[0],gsl_sf_bessel_J1(x),5.0e-3,"astep");
    t.test_rel(y[1],0.5*(gsl_sf_bessel_J0(x)-gsl_sf_bessel_Jn(2,x)),
               5.0e-3,"astep");
    t.test_rel(yerr[0],0.0,4.0e-8,"astep 3");
    t.test_rel(yerr[1],0.0,4.0e-8,"astep 4");
    t.test_gen(retx==0,"astep 5");
    k++;
}

```

```

}
cout << "Accuracy at end: "
      << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

// Solve using the O2scl initial value solver

cout << "Initial value solver: " << endl;
ode_iv_solve<double> ode4;
const int ngrid=101;
ovector xg(ngrid), yinit(2);
omatrix yg(ngrid,2);
for(i=0;i<ngrid;i++) xg[i]=((double)i)/10.0;
yinit[0]=0.0;
yinit[1]=0.5;
ode4.solve_grid(0.0,10.0,0.1,2,yinit,ngrid,xg,yg,alpha,od);

cout << " x          J1(calc)          J1(exact)          rel. diff." << endl;
for(i=1;i<ngrid;i+=10) {
    cout << xg[i] << " " << yg[i][0] << " "
          << gsl_sf_bessel_J1(xg[i]) << " ";
    cout << fabs((yg[i][0]-gsl_sf_bessel_J1(xg[i]))/
                  gsl_sf_bessel_J1(xg[i])) << endl;
    t.test_rel(yg[i][0],gsl_sf_bessel_J1(xg[i]),5.0e-7,"astep");
    t.test_rel(yg[i][1],0.5*(gsl_sf_bessel_J0(xg[i])-
                             gsl_sf_bessel_Jn(2,xg[i])),5.0e-7,"astep 2");
}
cout << "Accuracy at end: "
      << fabs(yg[ngrid-1][0]-gsl_sf_bessel_J1(xg[ngrid-1]))/
      gsl_sf_bessel_J1(xg[ngrid-1]) << endl;
cout << endl;

cout.unsetf(ios::showpos);
t.report();

return 0;
}
// End of example

```

1.32.9 Simulated annealing example

```

/* Example: ex_anneal.cpp
-----
An example to demonstrate minimization by simulated annealing
*/

#include <iostream>
#include <cmath>
#include <gsl/gsl_sf_bessel.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/multi_funct.h>
#include <o2scl/funct.h>
#include <o2scl/gsl_anneal.h>
#include <o2scl/test_mgr.h>

using namespace std;
using namespace o2scl;

// A simple function with many local minima. A "greedy" minimizer
// would likely fail to find the correct minimum.
double function(size_t nvar, const ovector_base &x, int &vp) {
    double ret, a, b;
    a=(x[0]-2.0);
    b=(x[1]+3.0);
    return -gsl_sf_bessel_J0(a)*gsl_sf_bessel_J0(b);
}

int main(int argc, char *argv[]) {

```

```

test_mgr t;
t.set_output_level(1);

cout.setf(ios::scientific);

gsl_anneal<int,multi_funct<int> > ga;
double result;
ovector init(2);

multi_funct_fptr_noerr<int> fx(function);

ga.ntrial=100;
ga.verbose=1;
ga.tolx=1.0e-6;

int vpx=0;

// Choose an initial point at a local minimum away from
// the global minimum
init[0]=9.0;
init[1]=9.0;

// Perform the minimization
ga.mmin(2,init,result,vpx,fx);
cout << "x: " << init[0] << " " << init[1]
    << ", minimum function value: " << result << endl;
cout << endl;

// Test that it found the global minimum
t.test_rel(init[0],2.0,1.0e-2,"another test - value");
t.test_rel(init[1],-3.0,1.0e-2,"another test - value 2");
t.test_rel(result,-1.0,1.0e-2,"another test - min");

t.report();

return 0;
}
// End of example

```

1.32.10 Multidimensional integration example

```

/* Example: ex_minte.cpp
-----
An example to demonstrate multidimensional integration
*/

#include <o2scl/test_mgr.h>
#include <o2scl/multi_funct.h>
#include <o2scl/composite_inte.h>
#include <o2scl/gsl_inte_qng.h>
#include <o2scl/gsl_vegas.h>

/// For M_PI
#include <gsl/gsl_math.h>

using namespace std;
using namespace o2scl;

double test_fun(size_t nv, const ovector_base &x, void *&vp) {
    double y=1.0/(1.0-cos(x[0])*cos(x[1])*cos(x[2]))/M_PI/M_PI/M_PI;
    return y;
}

int main(void) {
    test_mgr t;
    t.set_output_level(1);

    cout.setf(ios::scientific);

```

```

double exact = 1.3932039296;
double res;

double err;

void *vpx=0;
gsl_vegas<void *,multi_funct<void *> > gm;
ovector a(3), b(3);
a.set_all(0.0);
b.set_all(M_PI);

multi_funct_fptr_noerr<void *> tf(test_fun);

gm.n_points=100000;
gm.minteg_err(tf,3,a,b,vpx,res,err);

cout << res << " " << exact << " " << (res-exact)/err << endl;
t.test_rel(res,exact,err*10.0,"O2scl");

t.report();

return 0;
}
// End of example

```

1.32.11 Contour lines example

```

/* Example: ex_contour.cpp
-----
Example for generating contour lines
*/

#include <iostream>
#include <o2scl/contour.h>
#include <o2scl/ovector_tlate.h>

using namespace std;
using namespace o2scl;

// A function defining the three-dimensional surface
// for which we want to compute contour levels
double fun(double x, double y) {
    return 15.0*exp(-pow(x-20.0,2.0)/400.0-pow(y-5.0,2.0)/25.0)
        +40.0*exp(-pow(x-70.0,2.0)/500.0-pow(y-2.0,2.0)/4.0);
}

// A function for outputting the data to cout
int print_data(int nx, int ny, ovector_base &x, ovector_base &y,
              omatrix_base &data);

// A function for printing the contour information to a file
int file_out(vector<contour_line> &conts, vector<contour_line> &conts2);

int main(void) {
    test_mgr t;
    t.set_output_level(1);

    cout.setf(ios::scientific);

    contour co;

    // Initialize the data

    ovector x(12), y(10);
    omatrix data(10,12);
    for(size_t i=0;i<10;i++) {
        y[i]=((double)i);
    }
}

```

```

}
for(size_t i=0;i<12;i++) {
    x[i]=((double)i)*((double)i);
}

for(size_t j=0;j<12;j++) {
    for(size_t k=0;k<10;k++) {
        data[k][j]=fun(x[j],y[k]);
    }
}
co.set_data(12,10,x,y,data);

// Print out the data

print_data(12,10,x,y,data);

// Set the contour levels

ovector levels(7);
levels[0]=5.0;
levels[1]=10.0;
levels[2]=15.0;
levels[3]=20.0;
levels[4]=25.0;
levels[5]=30.0;
levels[6]=35.0;

co.set_levels(7,levels);

// Compute the contours

vector<contour_line> conts;
co.calc_contours(conts);

// Print the contours to the screen and test to make sure
// that they match the requested level

size_t nc=conts.size();
for(size_t i=0;i<nc;i++) {
    cout << "Contour " << i << " at level " << conts[i].level << ":" << endl;
    size_t cs=conts[i].x.size();
    for(size_t j=0;j<cs;j++) {
        cout << "(" << conts[i].x[j] << ", " << conts[i].y[j] << ")" << endl;
        //t.test_rel(fun(conts[i].x[j],conts[i].y[j]),conts[i].level,
        //1.5e-1,"curve");
    }
    cout << endl;
}

// Refine the data using cubic spline interpolation

def_interp_mgr<ovector_const_view,cspline_interp> dim1;
def_interp_mgr<ovector_const_subvector,cspline_interp> dim2;
co.regrid_data(5,5,dim1,dim2);

// Recompute the contours

vector<contour_line> conts2;
co.calc_contours(conts2);

// Output the contour information to a file for the documentation
file_out(conts,conts2);

t.report();

return 0;
}
// End of example

```

1.32.12 Two-dimensional interpolation example

```

/* Example: ex_twod_intp.cpp
-----
A simple example for two-dimensional interpolation using
the twod_intp class.
*/

#include <o2scl/twod_intp.h>
#include <o2scl/test_mgr.h>

using namespace std;
using namespace o2scl;

// A function for filling the data and comparing results
double f(double x, double y) {
    return pow(sin(0.1*x+0.3*y),2.0);
}

int main(void) {
    int i,j;

    test_mgr t;
    t.set_output_level(1);

    // Create the sample data

    ovector x(3), y(3);
    omatrix data(3,3);

    cout.setf(ios::scientific);

    // Set the grid
    x[0]=0.0;
    x[1]=1.0;
    x[2]=2.0;
    y[0]=3.0;
    y[1]=2.0;
    y[2]=1.0;

    // Set and print out the data
    cout << endl;
    cout << "          x | ";
    for(i=0;i<3;i++) cout << x[i] << " ";
    cout << endl;
    cout << " y          |" << endl;
    cout << "-----|-----";
    for(i=0;i<3;i++) cout << "-----";
    cout << endl;
    for(i=0;i<3;i++) {
        cout << y[i] << " | ";
        for(j=0;j<3;j++) {
            data[i][j]=f(x[j],y[i]);
            cout << data[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;

    // Perform the interpolation

    cout << "x          y          Calc.          Exact" << endl;

    twod_intp ti;

    // Interpolation, x-first
    double tol=0.05;
    double tol2=0.4;

    ti.set_data(3,3,x,y,data,true);

```

```

double x0, y0, x1, y1;

x0=0.5; y0=1.5;
cout << x0 << " " << y0 << " "
    << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

x0=0.99; y0=1.99;
cout << x0 << " " << y0 << " "
    << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

x0=1.0; y0=2.0;
cout << x0 << " " << y0 << " "
    << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

cout << endl;

// Interpolation, y-first

ti.set_data(3,3,x,y,data,false);

x0=0.5; y0=1.5;
cout << x0 << " " << y0 << " "
    << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

x0=0.99; y0=1.99;
cout << x0 << " " << y0 << " "
    << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

x0=1.0; y0=2.0;
cout << x0 << " " << y0 << " "
    << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

cout << endl;

t.report();
return 0;
}
// End of example

```

1.32.13 ublas vector example

```

/* Example: ex_ublas.cpp
-----
This gives an example of the how one can use the ublas vector types
in an O2scl interpolation class and ublas vector and matrix types
in an O2scl equation solver class
*/

#include <iostream>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <o2scl/ovector_tlate.h>
#include <o2scl/omatrix_tlate.h>
#include <o2scl/vec_arith.h>
#include <o2scl/mm_funct.h>
#include <o2scl/gsl_mroot_hybrids.h>
#include <o2scl/test_mgr.h>
#include <o2scl/interp.h>

// Some convenient typedefs
typedef boost::numeric::ublas::vector<double> ubvector;
typedef boost::numeric::ublas::matrix<double> ubmatrix;

using namespace std;
using namespace o2scl;

// An allocation object for ublas vectors

```



```

class ubvector_alloc {
public:
    // Ensure space in \c u for \c i elements
    void allocate(ubvector &u, size_t i) { u.resize(i); }
    // Free memory
    void free(ubvector &u) { }
};

// An allocation object for ublas matrices
class ubmatrix_alloc {
public:
    // Ensure space in \c u for \c i elements
    void allocate(ubmatrix &u, size_t i, size_t j) { u.resize(i,j); }
    // Free memory
    void free(ubmatrix &u, size_t i) { }
};

// A class with a function to solve
class cl {

public:

    int function(size_t nv, const ubvector &x, ubvector &y, int &pa) {
        y[0]=sin(x[1]-0.2);
        y[1]=sin(x[0]-0.25);
        return 0;
    }

};

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

    test_mgr t;
    t.set_output_level(1);

    cl acl;

    // Create some data to be interpolated
    ubvector x(10), y(10);
    for(size_t i=0;i<10;i++) {
        x[i]=i;
        y[i]=sin(x[i]);
    }

    // Create the interpolation object
    def_interp_mgr<ubvector,cspline_interp> dim;
    o2scl_interp_vec<ubvector,ubvector,ubvector_alloc> ip(dim,10,x,y);

    // Try the interpolation
    cout.setf(ios::showpos);
    cout << "Interpolation: " << endl;
    for(double z=0.41;z<8.0;z+=0.8) {
        cout << z << " " << ip.interp(z) << " " << sin(z) << endl;
        t.test_rel(ip.interp(z), sin(z), 1.0e-2, "Interpolation");
    }
    cout << endl;
    cout.unsetf(ios::showpos);

    /*
    // Create a new function object and solver with the
    // ublas vector and matrix types.
    mm_funct_mfp_ptr<cl,int,ubvector> f1(&acl,&cl::function);
    //gsl_mroot_hybrids<int,mm_funct<int,ubvector>,ubvector,ubvector,
    //ubvector_alloc,ubmatrix,ubmatrix,ubmatrix_alloc> gmr;

    ubvector sol(2);
    sol[0]=0.5;
    sol[1]=0.5;
    int pa;

```

```
//gmr.msolve(2,sol,pa,f1);
cout << "Solver: " << sol[0] << " " << sol[1] << endl;
cout << endl;
t.test_rel(sol[0],0.25,1.0e-6,"Solution 1.");
t.test_rel(sol[1],0.2,1.0e-6,"Solution 2.");
*/

t.report();
return 0;
}
// End of example
```

1.33 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 some of the most "top-level" header files and their associated dependencies within `O2scl` (there are other dependencies on GSL and the C standard library not listed here).

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

The use of templates

Templates are used extensively, and 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.

1.33.1 Error handling

Thread safety

Two approaches to thread-safe error handling which are worth comparing: the first is GSL which uses return codes and global function for an error handler, and the second is the Math/Special Functions section of Boost, which uses a separate policy type for

each function. One issue is thread safety: the GSL approach is thread safe only in the sense that one can in principle use the return codes in different threads to track errors. What one cannot do in GSL is use different user-defined error handlers for different threads. The Special Functions library allows one to choose a different Policy for every special function call, and thus allows quite a bit more flexibility in designing multi-threaded error handling.

1.33.2 Vector design

O₂scl vector and matrix types are a hybrid approach: creating objects compatible with GSL, while providing syntactic simplicity and object-oriented features common to C++ vector classes. In terms of their object-oriented nature, they are not as elegant as the `ublas` vector types from `ublas`, but for many applications they are also faster (and they are always at least as fast).

However, ensuring const-correctness makes the design a bit thorny, and this is still in progress.

1.33.3 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, 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_tlate.h](#).

1.33.4 Define constants and macros

There are a couple define constants and macros that O₂scl understands, they are all in upper case and begin with the prefix `O2SCL_`.

Range-checking for arrays and matrices is turned on by default, but can be turned off by defining `O2SCL_NO_RANGE_CHECK` during the initial configuration of the library. To see how the library was configured at runtime, use the [lib_settings](#) class.

There are several macros for error handling defined in [err_hnd.h](#), and several for vector/matrix arithmetic in [vec_arith.h](#).

There is a define constant `O2SCL_NO_SYSTEM_FUNC` which permanently disables the shell command '!' in `cli` (when the constant is defined, the shell command doesn't work even if `cli::shell_cmd_allowed` is `true`).

The constant `O2SCL_DATA_DIR` is defined internally to provide the directory which contains the O₂scl data files. After installation, this can be accessed in [lib_settings](#).

All of the header files have their own define constant of the form `O2SCL_HEADER_FILE_NAME` which ensures that the header file is only included once.

Finally, I sometimes comment out sections of code with

```
#ifdef O2SCL_NEVER_DEFINED
...
#endif
```

This constant should not be defined by the user as it will cause compilation to fail.

1.33.5 Global objects

There are three global objects that are created in `libo2scl`:

- `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 to be avoided.

1.33.6 Thread safety

Most of 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`.

1.33.7 Documentation design

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.

1.33.8 Copyright notices

For files where it is appropriate to do so, I have followed the prescription suggested in <http://lists.gnu.org/archive/html/help-gsl/2008-11/msg00017.html> retaining the `GSL` copyright notices and putting the `O2scl` notices at the top. `CERNLIB` has no such standard, but their licensing information is outlined at <http://cernlib.web.cern.ch/cernlib/conditions.html>.

1.33.9 Design plans

Exception classes:

Eventually I'd like to create a sensible hierarchy of exception classes to be thrown by the handler if the user would prefer (this is already in progress).

Boost and linear algebra:

I would like to ensure this class is compatible with `boost`, and start integrating things accordingly. IMHO object-oriented linear algebra is in a rather sad state at the moment. `uBlas` and `MTL` are both promising, however, and I'd like to start implementing some sort of compatibility with `uBlas` vectors and matrices soon. The `uBlas` documentation is pretty sparse, but that's the pot calling the kettle a cheap piece of metal.

Other Improvements:

I'm particularly interested in improving the ODE and fitting classes, as well as updating the BFGS2 minimizer. Of course, more examples and better documentation are also a must.

Algorithms to include

- Method of lines for PDEs
- Some of the MESA interpolation routines.
- C++ translation of MINUIT (done already by ROOT, but quite difficult).
- Creating closed regions from [contour](#) lines (I have no idea how to do this at the moment, though I'm sure someone has solved this problem already somewhere.)

Complex numbers

I'm not sure where to go with complex numbers. My guess is that `std::complex` is not significantly slower (or is faster) than `gsl_complex`, but it would be good to check this. Then there's the C99 standard, which is altogether different. Unfortunately the interfaces may be sufficiently different that it's impossible to make templated classes which operate on generic complex number types.

I/O

The I/O classes are (admittedly) not structured very well, and there are a lot of things which could be done there. There are a lot of alternative solutions out there, but most of them work on binary file formats (which I find difficult to use).

- On a shorter time scale I'd like to replace the inner workings of the [collection](#) class with a mechanism using `boost::any`, which will avoid the silly `void *`s all over the place. This is already in progress.
- On a longer time scale, I'd like to replace the file format I/O classes with something using Boost pipes.

1.34 License Information

O2scl (as well as CERNLIB and the Gnu Scientific Library (GSL)) is licensed under version 3 of 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.

1.35 Acknowledgements

I would like to thank the creators of GSL and Doxygen for their excellent work! Thanks also to Julien Garaud for contributing the ODE multishooting class and for the `ex_hydrogen` example.

1.36 Bibliography

Some of the references which contain links should direct you to the work referred to directly through dx.doi.org.

Bader83: G. Bader and P. Deuffhard, *Numer. Math.* **41** (1983) 373.

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

Cash90: Cash, J.R., Karp, A.H., *ACM Transactions of Mathematical Software*, vol. 16 (1990) 201-222.

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

Hairer00: Hairer, E., Norsett S.P., Wanner, G. *Solving ordinary differential equations I, Nonstiff Problems*, 2nd revised edition, Springer, 2000.

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

Lepage78: G. P. Lepage, originally described in *J. Comp. Phys.* **27** (1978) 192.

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' and M.Y. Cosnard, *ACM Trans. Math. Software*, **5** (1979) 64-85.

More80: J.J. More' and M.Y. Cosnard, *Algorith 554 BRENTM*, *Collected Algorithms from CACM* (1980).

Nelder65: Nelder, J.A., Mead, R., *Computer Journal* **7** (1965) pp. 308-313.

Press90: W.H. Press, G.R. Farrar, "Recursive Stratified Sampling for Multidimensional Monte Carlo Integration", *Computers in Physics*, v4 (1990), pp. 190-195.

Prince81: P.J. Prince and J.R. Dormand *J. Comp. Appl. Math.*, **7** (1981) 67.

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

2 Download O2scl

The current version is 0.904. The source distribution can be obtained from

- http://sourceforge.net/project/showfiles.php?group_id=206918

You may also download O₂scl from version from the Subversion repository. To obtain the bleeding-edge developer version, use something like

```
svn co https://o2scl.svn.sourceforge.net/svnroot/o2scl/trunk o2scl
```

The checkout process is a bit more time consuming than directly downloading the source distribution.

Version 0.904 corresponds with version 40 in the svn repository.

Version 0.903 does not exist (this number was skipped to help coordinate numbering with `O2scl_ext`).

Version 0.902 corresponds with version 37 in the svn repository.

Version 0.901 corresponds with version 31 in the svn repository.

Version 0.9 corresponds with version 26 in the svn repository.

Version 0.806 was a quick release to fix some issues with compilations in Mac OS X and does not correspond exactly with a version in the repository.

Version 0.805 corresponds with version 16 in the svn repository.

3 Ideas for future development

Class `akima_interp` It appears that the `interp()` function below searches for indices slightly differently than the original GSL `eval()` function. This should be checked, as it might be slightly non-optimal in terms of speed (shouldn't matter for the accuracy).

Class `anneal_mt` There may be a good way to remove the function indirection here to make this class a bit faster.

Class `array_alloc` Might it be possible to rework this so that it does range checking and ensures that the user doesn't try to allocate more or less space? I.e. `array_alloc<double[2]>` complains if you try an `allocate(x,3)`?

Class `array_const_subvector` Make the member data truly const and remove the extra typecast.

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. (I think I might have removed the acceleration)

Class `cern_adapt`

- Allow user to set the initial segments?
- It might be interesting to directly compare the performance of this class to `gsl_inte_qag`.

Class `cern_gauss` Allow user to change `cst`?

Global `cern_gauss::integ(func_t &func, double a, double b, param_t &pa)` Modify this to include iteration count information as done in `cern_cauchy`

Class `cern_mroot` Modify this so it handles functions which return non-zero values.

Class `cern_mroot` Move some of the memory allocation out of `msolve()`

Class `cern_mroot` Give the user access to the number of function calls

Class `cern_mroot` Rename `nier6`, `nier7`, and `nier8` to something sensible.

Class `cern_mroot_root` Double-check this class to make sure it cannot fail while returning 0 for success.

Global `cern_mroot_root::eps` This number should probably default to one of the GSL tolerances.

Class `cli` Include a "remove command" function

Class `cli` A replace command function, there's already some code in `cli.cpp` for this.

Class `cli` There's some code duplication between `comm_option_run()` and `run_interactive()`

Class `cli` Allow the user to set the tilde string

Class `columnify` Move the `screenify()` functionality from `misc.h` into this class?

Class `comp_gen_inte` Provide an example of usage for this class.

Class `composite_inte` Create a function to set an entire array of one-dimensional integration objects at once

Class `contour` Rewrite the code which adjusts the `contour` levels to ensure contours don't go through the data to adjust the internal copy of the data instead.

Class `contour` It would be nice to have a function which creates a set of closed regions to fill which represent the data. However, this likely requires a completely new algorithm, because it's not easy to simply close the contours already generated by the `calc_contours()` function. There are, for example, several cases which are difficult to handle, such as filling a region in between several closed contours.

Global `contour::get_data(size_t &size_x, size_t &size_y, ovector * &x_fun, ovector * &y_fun, omatrix * &udata)` There is probably a better way than returning pointers to the internal data.

Class `contour_line` Write an I/O object for `contour` lines

Class `contour_line` Make this a subclass of `contour`.

Class `convert_units` Ideally, a real C++ API for the GNU units command would probably be better.

Class `cspline_interp` Could use `O2scl` 's native tridiagonal routines instead of calling the GSL ones.

Class `deriv` Improve the methods for second and third derivatives

Class `edge_crossings` Write an I/O object for edge crossings

Class `edge_crossings` Make this a subclass of `contour` .

Class `err_base` There may be an issue associated with the string manipulations causing errors in the error handler.

Class `file_detect` Allow the user to specify the compression commands in `configure`, or at least specify the path to `gzip`, `bzip2`, etc.

Class `file_detect` Use the boost pipes facility instead.

Class `format_float` Implement padding with zeros

Class `format_float` Separate out decimal point and make it separate.

Class `gen_test_number` Document what happens if `t0t` is small

Class `gsl_anneal` There's `x0`, `old_x`, `new_x`, `best_x`, and `x`? There's probably some duplication here which could be avoided.

Class `gsl_anneal` • Implement a more general simulated annealing routine which would allow the solution of discrete problems like the Traveling Salesman problem.

Class `gsl_bsimp` I don't like setting `yerr` to `GSL_POSINF`, there should be a better way to force an adaptive stepper which is calling this stepper to readjust the stepsize.

Class `gsl_bsimp` Some of these functions can be moved out of this header file

Class `gsl_bsimp` Rework internal arrays as `uvector`s?

Class `gsl_bsimp` The function `step_local()` is actually its own ODE stepper and could be reimplemented as an object of type `odestep`

Class `gsl_deriv` Include the forward and backward GSL derivatives

Global `gsl_inte_kronrod::gsl_integration_qk_o2scl(func_t &func, const int n, const double xgk[], const double wg[], const double wkgk[],`
This function, in principle, could be replaced with a generic integration pointer.

Class `gsl_inte_qawc` Make `cern_cauchy` and this class consistent in the way which they require the user to provide the denominator in the integrand

Class `gsl_inte_qng` Compare directly with GSL as is done in `gsl_inte_qag_ts`.

Global `gsl_inte_qng::integ_err(func_t &func, double a, double b, param_t &pa, double &res, double &err2)` Allow user to change 0.5e28

Class `gsl_inte_singular` Some of the functions inside this class could be moved out of header files?

Global `gsl_inte_singular::qags(func_t &func, const int qn, const double xgk[], const double wg[], const double wkg[], double fv1[], double &res, double &err2)` Remove goto statements?

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. Some work on this front is already in `gsl_inte_qag_b.h`.

Global `gsl_mmin_base::minimize(const gsl_vector *x, const gsl_vector *xp, double lambda, double stepa, double stepb, double stepc, double &res, double &err2)` Remove goto statements

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::allocate(size_t n)` Use a `gsl_alloc_arrays()` like function for this (but keep in mind these are `calloc`, not `malloc` statements)

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_wrapper` There's a bit of extra vector copying here which could potentially be avoided.

Class `gsl_mroot_hybrids` It's kind of strange that `set()` sets `jac_given` to false and `set_de()` has to reset it to true. Can this be simplified?

Class `gsl_quartic_real` Optimize value of `cube_root_tol` and compare more clearly to `gsl_quartic_real2`

Class `gsl_quartic_real2` Optimize value of `cube_root_tol` and compare more clearly to `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_root_stef` There's some extra copying here which can probably be removed.

Class `gsl_root_stef` Compare directly to GSL.

Class `gsl_series` Convert to use a more general vector

Class `gsl_vegas` Could convert bins and boxes to a more useful structure

Class `gsl_vegas` The testing file calls the error handler on the `composite_inte` section. Fix this.

Class `lanczos` The function `eigen_tdiag()` automatically sorts the eigenvalues, which may not be necessary.

Class `lanczos` Do something better than the naive matrix-vector product?

Class `linear_solver` The test code uses a Hilbert matrix, which is known to be ill-conditioned, especially for the larger sizes. This should probably be changed.

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?

Class `nonadapt_step` Modify so that memory allocation/deallocation is only performed when necessary

Global `o2scl_hybrid_state_t::allocate(size_t n)` Convert to using `gsl_alloc_arrays()`

Class `ode_bv_shoot` Create a solution `table` as in `ode_iv_solve`

Class `ode_iv_solve` Add error information

Global `ode_iv_solve::solve_final_value_derivs(double x0, double x1, double h, size_t n, vec_t &ystart, vec_t ¥d, vec_t &dydx_start, v`
It looks like the `x0<x1` code and the `x1<x0` code is the same?

Global `ode_iv_solve::solve_table(double x0, double x1, double h, size_t n, vec_t &ystart, size_t &nsol, vec_t &xsol, mat_t &ysol, param_`
Should we give the option not to call the error handler in case running out of space in the `table`?

Global `ode_iv_solve::solve_table(double x0, double x1, double h, size_t n, vec_t &ystart, size_t &nsol, vec_t &xsol, mat_t &ysol, param_`
Consider modifying so that this can handle tables which are too small by removing half the rows and doubling the stepsize. Alternatively, make a function which accepts allocation objects and can re-allocate the vector and matrix space if required? (This latter option might be hard for arrays since they are really of fixed size.) A final alternative is to offer an `ovector/omatrix` interface which does the reallocation automatically if necessary.

Global `ode_iv_solve::solve_table(double x0, double x1, double h, size_t n, vec_t &ystart, size_t &nsol, vec_t &xsol, mat_t &ysol, param_`
 Some copying is done here and it would be nice if that could be avoided.

Class `ovector` Consider making `allocate()` and `free()` functions private for this class?

Class `ool_constr_mmin` Finish `mmin()` interface

Class `other_todos_and_bugs` • Fix the PNG images so that they're smaller and repair the transparency issue (probably can be done just using different arguments to 'convert' in the pngfix target)

- Make sure we have a `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 `other_todos_and_bugs` Consider breaking documentation up into sections?

Class `ovector_alloc` Could (or should?) the `allocate()` functions be adapted to return an integer error value?

Class `ovector_const_reverse_tlate` I think that maybe in order to ensure that this isn't created from an already reversed vector, this class has to be separated from the hierarchy altogether. However, I think this might break the smart interpolation stuff.

Global `ovector_const_view_tlate::norm() const` Move this function outside the vector template since it doesn't really work with integers.

Class `pinside` The `inside()` functions actually copy the points twice. This can be made more efficient.

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 `planar_intp` Instead of comparing `den` with zero, add the option of comparing it with a small number.

Class `rnga` Consider some analog of the GSL function `gsl_rng_uniform_pos()`, i.e. as used in the GSL Monte Carlo classes.

Global `rnga::clock_seed()` Figure out a better way of computing a random seed in a platform-independent way.

Class `root` Maybe consider allowing the user to specify the stream to which 'verbose' information is sent.

Class `root_de` At the moment, the functions `solve()` and `solve_bkt()` are not implemented for derivative solvers.

Global `search_vec::ordered_lookup(const double x0, size_t n, const vec_t &x)` This is not as efficient as it could be, as it just uses the `find_interval` functions and then adjusts the answer at the end if necessary.

Class `simple_jacobian` GSL-1.10 updated `fdjac.c` and this update could be implemented below.

Class `smart_interp` Change the determination of the array is increasing to be a bit more intelligent.

Class `smart_interp_vec` Properly implement handling of non-monotonic regions in the derivative functions as well as the interpolation function.

Class `table` Rewrite the `table::create_array()` and `table::insert_data()` functions for generic vector type

Class `table` Be more restrictive about allowable column names?

Class `table` Re-implement user-owned columns, and handle automatic resizing appropriately.

Class `table` Return the empty column in the `operator[]` functions as is done for the `get_column()` functions.

Class `table` A "delete rows" method to delete a range of several rows

Class `table` The present structure, `std::map<std::string, col, string_comp> atree` and `std::vector<aiter> alist`; could be replaced with `std::vector<col> list` and `std::map<std::string, int> tree` where the map just stores the index of the the column in the list

Class `table3d` Improve interpolation and derivative caching

Class `table3d` Make a 'const' version of the interpolation functions

Class `tensor` Could implement arithmetic operators + and - and some different products.

Class `tensor` Add slicing to get `ovector` or `omatrix` objects

Class `tensor_grid` Only allocate space for grid if it is set

Class `tensor_grid` Could implement arithmetic operators + and - and some different products.

Global `tensor_grid::tensor_grid(size_t rank, size_t *dim)` Create a "tensor_grid1" for completeness.

Global `tensor_grid::interpolate(double *vals)` It should be straightforward to improve the scaling of this algorithm significantly by creating a "window" of local points around the point of interest. This could be done easily by constructing an initial subtensor.

Global `tensor_grid::set_grid(double **val)` Define a more generic interface for matrix types

Class `test_mgr` `test_mgr::success` and `test_mgr::last_fail` should be protected, but that breaks the `operator+()` function. Can this be fixed?

Class `twod_intp` Could also include mixed second/first derivatives: `deriv_xxy` and `deriv_xyy`.

Class `twod_intp` Implement an improved caching system in case, for example `xfirst` is true and the last interpolation used the same value of `x`.

Class `uvector_const_view_flate` Could allow user-defined specification of restrict keyword

Class `uvector_cx_view_flate` Write `lookup()` method, and possibly an `erase()` method.

File `cblas_base.h` Convert to `size_t` and add float and complex versions

Global `dnrm2_subcol` Could be made more efficient with a matrix-col like object

Global `daxpy_hv_sub` Implement explicit loop unrolling

Global `ddot_hv_sub` Implement explicit loop unrolling

File `collection.h` Figure out what to do with templated matrix and vector output. What might be ideal is a "vector_io" type which will input or output a generic vector of any type.

Global `matrix_out` If all of the matrix elements are positive integers and scientific mode is not set, then we can avoid printing the extra spaces.

File `cx_arith.h` Define operators with assignment for complex + double?

Global `O2SCL_ASSERT` Make this consistent with `assert()` using `NDEBUG`?

Global `LU_decomp` The "swap rows `j` and `i_pivot`" section could probably be made more efficient using a "matrix_row"-like object.

Global **LU_invert** could rewrite to avoid `mat_col_t`

Global **gsl_alloc_arrays** Fix so that `__FILE__` and `__LINE__` are implemented here.

File **omatrix_tlate.h** The `xmatrix` class demonstrates how `operator[]` could return an `ovector_array` object and thus provide more bounds-checking. This would demand including a new parameter in **omatrix_view_tlate** which contains the vector type.

Global **operator<<** This assumes that scientific mode is on and `showpos` is off. It'd be nice to fix this.

File **ovector_tlate.h** Clean up maybe by moving, for example, `ovector` reverse classes to a different header file

File **poly.h** The quartics are tested only for `a4=1`, which should probably be generalized.

Global **solve_tridiag_sym** Convert into class for memory management and combine with other functions below

Global **operator<<** Make this function work even when scientific mode is not on, either by converting to scientific mode and converting back, or by leaving scientific mode off and padding with spaces

Global **operator<<** This assumes that scientific mode is on and `showpos` is off. It'd be nice to fix this.

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?

Global **matrix_cx_copy_gsl** At present this works only with complex types based directly on the GSL complex format. This could be improved.

Global **vector_cx_copy_gsl** At present this works only with complex types based directly on the GSL complex format. This could be improved.

4 Todo List

Class **bin_size** This class is not working yet.

Class **cli** There are some fixme entries in `cli.cpp` associated with when `cop` is zero

Class **cli** Only add `get()` and `set()` commands if some parameters are set

Class **cli** Warn in `run_interactive()` when extra parameters are given

Global `cli::cli_gets(const char *c)` Should this be protected?

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 `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::calc_vector(double x, double x0, double dx, size_t nx, const vec_t &y)` Document or change the value of 100.0 which appears here

Global `eqi_deriv::deriv_vector(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_bsimp` Ensure error handling is sensible if the "derivs" or `jacobian` functions return a non-zero value

Class `gsl_bsimp` Create an example with a stiff diff eq. which requires this kind of stepper

Class `gsl_fit` Properly generalize other vector types than `ovector_base`

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_qag` Verbose output has been setup for this class, but this needs to be done for some of the other GSL-like integrators

Class `gsl_inte_qagiu` I had to add extra code to check for non-finite values for some integrations. This should be checked.

Class `gsl_inte_qags` The convergence errors need to be handled correctly in function `qags` in `gsl_inte_qag_b.h`.

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_qawf_sin](#) Improve documentation a little

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_qawo_sin](#) Improve documentation

Class [gsl_inte_qaws](#) Finish this!

Class [gsl_inte_qng](#) Shouldn't feval be last_iter ?

Class [gsl_inte_table](#) Make the workspace size protected

Class [gsl_miser](#) The testing file calls the error handler on the [composite_inte](#) section. Fix this.

Class [gsl_vegas](#) Need to double check that the verbose output is good for all settings of verbose.

Class [gsl_vegas](#) BINS_MAX and bins_max are somehow duplicates. Fix this.

Class [io_base](#) Should the remove() functions be moved to class [collection](#)?

Class [multi_min_fix](#) Generalize to all vector types

Class [multi_min_fix](#) Generalize to minimizers which require derivatives

Class [o2scl_interp](#) Make sure the min size is checked on both 'itp' and 'ritp'.

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 [o2scl_interp_vec](#) Specify a [base_interp_mgr](#) object instead of [base_interp](#) objects

Class [ode_bv_multishoot](#) Improve documentation a little and create testing code

Class [ode_it_solve](#) Implement as a child of [ode_bv_solve](#) ?

Class [ode_it_solve](#) Max and average tolerance?

Class [ode_it_solve](#) partial correction option?

Global `ode_iv_solve::solve_final_value_derivs(double x0, double x1, double h, size_t n, vec_t &ystart, vec_t ¥d, vec_t &dydx_start, v`
 Add error information

Global `ode_iv_solve::solve_final_value_derivs(double x0, double x1, double h, size_t n, vec_t &ystart, vec_t ¥d, vec_t &dydx_start, v`
 At present, `dydx_start` is computed, but it should probably be assumed that the user will provide this.

Global `ode_iv_solve::solve_grid_derivs(double x0, double x1, double h, size_t n, vec_t &ystart, size_t nsol, vec_t &xsol, mat_t &ysol, ma`
 Add error information

Class `ool_constr_mmin` Implement automatic computations of `gradient` and Hessian

Class `ool_constr_mmin` Construct a non-trivial example for the "examples" directory

Global `ool_constr_mmin::mmin(size_t nvar, vec_t &xx, double &fmin, param_t &pa, func_t &ff)` Need to finish this function somehow since it's pure virtual in `multi_min`.

Class `ool_mmin_pgrad` Complete the `mmin()` interface with automatic `gradient`

Class `ool_mmin_pgrad` Replace the explicit norm computation below with the more accurate `dnrm2` from `linalg`

Class `other_todos_and_bugs`

- The `o2scl-test` and `o2scl-examples` targets require `grep`, `awk`, `tail`, `cat`, and `wc`. It would be good to reduce this list to ensure better compatibility.
- More examples and benchmarks
- There are a couple classes which Doxygen doesn't yet parse properly for the class hierarchy: `gsl_root_stef`, and `ool_mmin_gencan`. Also should double check `smart_interp`, etc.
- Make sure `abs(-double)` isn't allowed by the `O2scl` and `GSL` headers with the correct flags

Class `other_todos_and_bugs` At present, the testing code assumes that shared libraries are installed. Can this be improved? (12/3/08 - I'm not sure what the status is on this...this should be checked.)

Class `other_todos_and_bugs` Make sure default template parameters are documented in each class, and make sure default template parameters exist where they should.

Class `ovector_cx_tlate` Add `subvector_stride`, `const_subvector_stride`

Class `pointer_alloc` Call error handler appropriately here

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 `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`

The error handling is a bit off here, as it can return a non-zero value even with there is no real "error". We should just make a new bool reference paramter.

Global `smart_interp::interp(const double x0, size_t n, const vec_t &x, const vec_t &y)` After calling `find_subset`, I think we might need to double check that `nn` is larger than the minimum interpolation size.

Class `table_units` Make `table` methods virtual? (not necessary yet since `delete_column()` isn't referred to internally)

Class `tensor` More complete testing.

Class `text_out_file` Test output with `<`'s and `>`'s and document this

Class `text_out_file` Document the difference between `flush()` and `end_line()`

Global `text_out_file::text_out_file(std::ostream *out_file, int width=80, bool bracket_objs=true)` Ensure streams are not opened in binary mode for safety.

File `cx_arith.h` Ensure all the functions are tested.

Global `solve_cyc_tridiag_sym` Put reference in correctly.

File `user_io.h` Do input for array and 2d array objects, finish output functions. Make a macro for I/O for doubles, ints and other objects w/o a `type()` function.

Global `o2scl_output` Do input for array and 2d array objects

Global `o2scl_output` Do input for array and 2d array objects

Global `o2scl_output` Do input for array and 2d array objects

File `vec_arith.h` Properly document the operators defined as macros

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 `gsl_mmin_simp2` This class doesn't work at the moment, and it's not yet clear if this bug is limited to the O₂scl version or may also be present in GSL.

Class `other_todos_and_bugs` • 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 (1/4/09 - I'm not sure I remember why this is a problem, but it probably has to do with the detection of libraries which is done in configure.ac).

6 Things which need documentation

Class `gsl_bsimp` More detailed documentation

Class `gsl_inte_qag` Document use of last_iter

Class `gsl_miser` Document the fact that min_calls and min_calls_per_bisection need to be set beforehand

Class `gsl_miser` Document the member data

Class `gsl_vegas` Document the member data more carefully

Class `hybrids_base` Document the individual functions for this class, and possibly rename it.

Class `o2scl_hybrid_state_t` Improve the documentation in this class, and possibly rename it.

7 Namespace Documentation

7.1 `gsl_cgs` Namespace Reference

GSL constants in CGS units.

Variables

- const double [schwarzschild_radius](#) = 2.95325008e5
cm
- const double [speed_of_light](#) = 2.99792458e10
cm / s
- const double [gravitational_constant](#) = 6.673e-8
cm³ / g s²
- const double [plancks_constant_h](#) = 6.62606876e-27
g cm² / s
- const double [plancks_constant_hbar](#) = 1.05457159642e-27
g cm² / 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 / s²
- const double [electron_volt](#) = 1.602176462e-12
g cm² / s²
- 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 cm² / s²
- const double [boltzmann](#) = 1.3806503e-16
g cm² / K s²
- const double [bohr_magneton](#) = 9.27400899e-20
A cm².
- const double [nuclear_magneton](#) = 5.05078317e-23
A cm².
- const double [electron_magnetic_moment](#) = 9.28476362e-20
A cm².
- const double [proton_magnetic_moment](#) = 1.410606633e-22
A cm².
- const double [molar_gas](#) = 8.314472e7
g cm² / K mol s²
- const double [standard_gas_volume](#) = 2.2710981e4
cm³ / 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*²
 - 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](#) = 9.6485341472e4
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 / cm²
- const double [footlambert](#) = 1.07639104e-3
cd sr / cm²
- const double [curie](#) = 3.7e10
1 / s
- const double [roentgen](#) = 2.58e-7
A s / g.
- const double [rad](#) = 1e2
cm² / s²
- const double [solar_mass](#) = 1.98892e33
g
- const double [bohr_radius](#) = 5.291772083e-9
cm
- const double [newton](#) = 1e5
cm g / s²
- const double [dyne](#) = 1e0
cm g / s²
- const double [joule](#) = 1e7
g cm² / s²
- const double [erg](#) = 1e0
g cm² / s²
- const double [stefan_boltzmann_constant](#) = 5.67039934436e-5
g / K⁴ s³
- const double [thomson_cross_section](#) = 6.65245853542e-25
cm²

7.1.1 Detailed Description

GSL constants in CGS units.

The CGS units are given below each constant

7.2 gsl_cgsm Namespace Reference

GSL constants in CGSM units.

Variables

- const double [schwarzschild_radius](#) = 2.95325008e5
cm
- const double [speed_of_light](#) = 2.99792458e10
cm / s
- const double [gravitational_constant](#) = 6.673e-8
cm³ / g s²
- const double [plancks_constant_h](#) = 6.62606876e-27
g cm² / s
- const double [plancks_constant_hbar](#) = 1.05457159642e-27
g cm² / 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^2
- const double [electron_volt](#) = 1.602176462e-12
- $g \text{ cm}^2 / s^2$
- 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
- $g \text{ cm}^2 / s^2$
- const double [boltzmann](#) = 1.3806503e-16
- $g \text{ cm}^2 / K s^2$
- const double [bohr_magneton](#) = 9.27400899e-21
- $abamp \text{ cm}^2$
- const double [nuclear_magneton](#) = 5.05078317e-24
- $abamp \text{ cm}^2$
- const double [electron_magnetic_moment](#) = 9.28476362e-21
- $abamp \text{ cm}^2$
- const double [proton_magnetic_moment](#) = 1.410606633e-23
- $abamp \text{ cm}^2$
- const double [molar_gas](#) = 8.314472e7
- $g \text{ cm}^2 / K \text{ mol } s^2$
- const double [standard_gas_volume](#) = 2.2710981e4
- cm^3 / 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^2
- 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

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

7.2.1 Detailed Description

GSL constants in CGSM units.

The CGSM units are given below each constant

7.3 gsl_mks Namespace Reference

GSL constants in MKS units.

Variables

- const double `schwarzschild_radius` = 2.95325008e3
 m
- const double `speed_of_light` = 2.99792458e8
 m / s
- const double `gravitational_constant` = 6.673e-11
 $m^3 / \text{kg s}^2$
- const double `plancks_constant_h` = 6.62606876e-34
 $\text{kg m}^2 / \text{s}$
- const double `plancks_constant_hbar` = 1.05457159642e-34
 $\text{kg m}^2 / \text{s}$
- 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^2
- const double `electron_volt` = 1.602176462e-19
 $\text{kg m}^2 / \text{s}^2$
- 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
 $\text{kg m}^2 / \text{s}^2$
- const double `boltzmann` = 1.3806503e-23
 $\text{kg m}^2 / \text{K s}^2$
- 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
 $A\ m^2$.
- const double [proton_magnetic_moment](#) = 1.410606633e-26
 $A\ m^2$.
- const double [molar_gas](#) = 8.314472e0
 $kg\ m^2 / K\ mol\ s^2$
- const double [standard_gas_volume](#) = 2.2710981e-2
 m^3 / 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^2
- const double [acre](#) = 4.04685642241e3
 m^2
- const double [barn](#) = 1e-28
 m^2
- const double [liter](#) = 1e-3
 m^3
- const double [us_gallon](#) = 3.78541178402e-3
 m^3
- const double [quart](#) = 9.46352946004e-4
 m^3
- const double [pint](#) = 4.73176473002e-4
 m^3
- const double [cup](#) = 2.36588236501e-4
 m^3
- const double [fluid_ounce](#) = 2.95735295626e-5
 m^3
- const double [tablespoon](#) = 1.47867647813e-5
 m^3

- const double [teaspoon](#) = 4.92892159375e-6
 m^3
- const double [canadian_gallon](#) = 4.54609e-3
 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](#) = 2.77777777778e-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
 $kg\ m / A^2\ s^2$

7.3.1 Detailed Description

GSL constants in MKS units.

The MKS units are given below each constant

7.4 gsl_mkxa Namespace Reference

GSL constants in MKSA units.

Variables

- const double [schwarzschild_radius](#) = 2.95325008e3
m
- const double [speed_of_light](#) = 2.99792458e8
m / s
- const double [gravitational_constant](#) = 6.673e-11
m³ / kg s²
- const double [plancks_constant_h](#) = 6.62606876e-34
kg m² / s
- const double [plancks_constant_hbar](#) = 1.05457159642e-34
kg m² / 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 / s²
- const double [electron_volt](#) = 1.602176462e-19
kg m² / s²
- 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 m² / s²
- const double [boltzmann](#) = 1.3806503e-23
kg m² / K s²
- const double [bohr_magneton](#) = 9.27400899e-24
A m².
- const double [nuclear_magneton](#) = 5.05078317e-27
A m².
- const double [electron_magnetic_moment](#) = 9.28476362e-24
A m².
- const double [proton_magnetic_moment](#) = 1.410606633e-26
A m².
- const double [molar_gas](#) = 8.314472e0
kg m² / K mol s²
- const double [standard_gas_volume](#) = 2.2710981e-2
m³ / 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*²
- const double [acre](#) = 4.04685642241e3
- m*²
- const double [barn](#) = 1e-28
- m*²
- const double [liter](#) = 1e-3
- m*³
- const double [us_gallon](#) = 3.78541178402e-3
- m*³
- const double [quart](#) = 9.46352946004e-4
- m*³
- const double [pint](#) = 4.73176473002e-4
- m*³
- const double [cup](#) = 2.36588236501e-4
- m*³
- const double [fluid_ounce](#) = 2.95735295626e-5
- m*³
- const double [tablespoon](#) = 1.47867647813e-5
- m*³
- const double [teaspoon](#) = 4.92892159375e-6
- m*³
- const double [canadian_gallon](#) = 4.54609e-3
- 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](#) = 2.77777777778e-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 **lamert** = 1e4
 $cd\ sr / m^2$
- const double **footlamert** = 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
 $kg\ m / A^2\ s^2$

7.4.1 Detailed Description

GSL constants in MKSA units.

The MKSA units are given below each constant

7.5 gsl_num Namespace Reference

GSL numerical constants.

Variables

- 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
- const double **fine_structure** = 7.2973525376e-3
Fine structure constant (updated from <http://physics.nist.gov/cuu/Constants>).
- const double **avogadro** = 6.02214179e23
Avogadro's number (updated from <http://physics.nist.gov/cuu/Constants>).

7.5.1 Detailed Description

GSL numerical constants.

7.6 o2scl Namespace Reference

The main O₂scl namespace.

7.6.1 Detailed Description

The main O₂scl namespace.

By default, all O₂scl classes and functions which are not listed as being in one of O₂scl's smaller specialized namespaces are in this namespace. This namespace has been removed from the documentation to simplify the formatting.

This namespace documentation is in the file [src/base/lib_settings.h](#)

7.7 o2scl_cblas Namespace Reference

Namespace for O₂scl CBLAS function templates with operator[].

Enumerations

- enum **O2CBLAS_ORDER** { **O2cblasRowMajor** = 101, **O2cblasColMajor** = 102 }
Matrix order, either column-major or row-major.
- enum **O2CBLAS_TRANSPOSE** { **O2cblasNoTrans** = 111, **O2cblasTrans** = 112, **O2cblasConjTrans** = 113 }
Transpose operations.
- enum **O2CBLAS_UPLO** { **O2cblasUpper** = 121, **O2cblasLower** = 122 }
Upper- or lower-triangular.
- enum **O2CBLAS_DIAG** { **O2cblasNonUnit** = 131, **O2cblasUnit** = 132 }
Unit or generic diagonal.
- enum **O2CBLAS_SIDE** { **O2cblasLeft** = 141, **O2cblasRight** = 142 }
Left or right sided operation.

Functions

- template<class mat_t >
int **dgemm** (const enum **O2CBLAS_ORDER** Order, const enum **O2CBLAS_TRANSPOSE** TransA, const enum **O2CBLAS_TRANSPOSE** TransB, const int M, const int N, const int K, const double alpha, const mat_t &A, const mat_t &B, const double beta, mat_t &C)

Compute $y = \alpha \text{op}(A)x + \beta y$.

Standard BLAS functions

- template<class vec_t, class vec2_t >
void **daxpy** (const int N, const double alpha, const vec_t &X, vec2_t &Y)
Compute $y = \alpha x + y$.
- template<class vec_t, class vec2_t >
double **ddot** (const int N, const vec_t &X, const vec2_t &Y)
Compute $r = x \cdot y$.
- template<class vec_t >
void **dscal** (const int N, const double alpha, vec_t &X)
Compute $x = \alpha x$.
- template<class vec_t >
double **dnrm2** (const int N, const vec_t &X)
Compute the squared norm of the vector X.
- template<class mat_t, class vec_t >
int **dgemv** (const enum **O2CBLAS_ORDER** order, const enum **O2CBLAS_TRANSPOSE** TransA, const int M, const int N, const double alpha, const mat_t &A, const vec_t &X, const double beta, vec_t &Y)
Compute $y = \alpha \text{op}(A)x + \beta y$.
- template<class mat_t, class vec_t >
int **dtrsv** (const enum **O2CBLAS_ORDER** order, const enum **O2CBLAS_UPLO** Uplo, const enum **O2CBLAS_TRANSPOSE** TransA, const enum **O2CBLAS_DIAG** Diag, const int M, const int N, const mat_t &A, vec_t &X)
Compute $x = \text{op}(A)^{-1}x$.

Helper BLAS functions

- template<class vec_t, class vec2_t >
void **daxpy_subvec** (const int N, const double alpha, const vec_t &X, vec2_t &Y, const int ie)
Compute $x = \alpha x$ beginning with index ie and ending with index N-1.
- template<class vec_t, class vec2_t >
double **ddot_subvec** (const int N, const vec_t &X, const vec2_t &Y, const int ie)
Compute $r = x \cdot y$ beginning with index ie and ending with index N-1.
- template<class vec_t >
void **dscal_subvec** (const int N, const double alpha, vec_t &X, const int ie)
Compute $x = \alpha x$ beginning with index ie and ending with index N-1.
- template<class vec_t >
double **dnrm2_subvec** (const int N, const vec_t &X, const int ie)
Compute the squared norm of the vector X beginning with index ie and ending with index N-1.
- template<class mat_t >
double **dnrm2_subcol** (const mat_t &M, const size_t ir, const size_t ic, const size_t N)
Compute the squared norm of the last N rows of a column of a matrix.
- template<class mat_t, class mat_subcol_t >
double **dnrm2_subcol2** (const mat_t &M, const size_t ir, const size_t ic, const size_t N)
Desc.
- template<class mat_t >
void **dscal_subcol** (mat_t &A, const size_t ir, const size_t ic, const size_t n, const double alpha)
Compute $x = \alpha x$.
- template<class mat_t, class vec_t >
void **daxpy_hv_sub** (const int N, const double alpha, const mat_t &X, vec_t &Y, const int ie)
Compute $x = \alpha x$ for [householder_hv_sub\(\)](#).
- template<class mat_t, class vec_t >
double **ddot_hv_sub** (const int N, const mat_t &X, const vec_t &Y, const int ie)
Compute $r = x \cdot y$ for [householder_hv_sub\(\)](#).

7.7.1 Detailed Description

Namespace for O2scl CBLAS function templates with operator[].

7.7.2 Function Documentation

7.7.2.1 `void o2scl_cblas::daxpy_hv_sub (const int N, const double alpha, const mat_t & X, vec_t & Y, const int ie)` `[inline]`

Compute $x = \alpha x$ for [householder_hv_sub\(\)](#).

Used in [householder_hv_sub\(\)](#).

Idea for future

Implement explicit loop unrolling

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

7.7.2.2 `void o2scl_cblas::daxpy_subvec (const int N, const double alpha, const vec_t & X, vec2_t & Y, const int ie)` `[inline]`

Compute $x = \alpha x$ beginning with index `ie` and ending with index `N-1`.

Used in [householder_hv\(\)](#).

Definition at line 398 of file `cblas_base.h`.

7.7.2.3 `double o2scl_cblas::ddot_hv_sub (const int N, const mat_t & X, const vec_t & Y, const int ie)` `[inline]`

Compute $r = x \cdot y$ for [householder_hv_sub\(\)](#).

Used in [householder_hv_sub\(\)](#).

Idea for future

Implement explicit loop unrolling

Definition at line 665 of file `cblas_base.h`.

7.7.2.4 `double o2scl_cblas::ddot_subvec (const int N, const vec_t & X, const vec2_t & Y, const int ie)` `[inline]`

Compute $r = x \cdot y$ beginning with index `ie` and ending with index `N-1`.

Used in [householder_hv\(\)](#).

Definition at line 428 of file `cblas_base.h`.

7.7.2.5 `double o2scl_cblas::dnrm2_subcol (const mat_t & M, const size_t ir, const size_t ic, const size_t N)` `[inline]`

Compute the squared norm of the last `N` rows of a column of a matrix.

Given matrix `M`, this computes the norm of the last `N` rows of the column with index `ic`, beginning with the element with index `ir`. If the matrix `M` has `r` rows, and `c` columns, then the parameter `N` should be `r-ir`.

Used in [householder_transform_subcol\(\)](#).

Idea for future

Could be made more efficient with a matrix-col like object

Definition at line 526 of file `cblas_base.h`.

7.7.2.6 double o2scl_cblas::dnrm2_subvec (const int N , const vec_t & X , const int ie) [inline]

Compute the squared norm of the vector X beginning with index ie and ending with index $N-1$.

Used in [householder_transform\(\)](#).

Definition at line 480 of file cblas_base.h.

7.7.2.7 void o2scl_cblas::dscal_subcol (mat_t & A , const size_t ir , const size_t ic , const size_t n , const double $alpha$) [inline]

Compute $x = \alpha x$.

Used in [householder_transform_subcol\(\)](#).

Definition at line 602 of file cblas_base.h.

7.7.2.8 void o2scl_cblas::dscal_subvec (const int N , const double $alpha$, vec_t & X , const int ie) [inline]

Compute $x = \alpha x$ beginning with index ie and ending with index $N-1$.

Used in [householder_transform\(\)](#).

Definition at line 456 of file cblas_base.h.

7.8 o2scl_cblas_paren Namespace Reference

Namespace for O2scl CBLAS function templates with operator().

7.8.1 Detailed Description

Namespace for O2scl CBLAS function templates with operator().

This namespace contains an identical copy of all the functions given in the [o2scl_cblas](#) namespace, but perform array indexing with `operator()` rather than `operator[]`. See [o2scl_cblas](#) for the function listing and documentation.

7.9 o2scl_const Namespace Reference

O2scl constants.

Variables

- const double [pi](#) = $\text{acos}(-1.0)$
 π
- const double [pi2](#) = $\text{pi} * \text{pi}$
 π^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 `sin2_theta_weak` = 0.2224
 $\sin^2 \theta_W$
- const double `mev_kg` = 1.782661731e-30
1 MeV in kg
- const double `ev_mks` = 1.602176462e-19
1 eV in kg · m²/s² (Joules)
- const double `mev_cgs` = 1.60217733e-6
1 MeV in g · cm²/s² (ergs)
- const double `boltzmann_mev_K` = 8.617342e-11
1 MeV in Kelvin

From <http://physics.nist.gov/cuu/Constants>

- const double `hc_mev_fm` = 197.3269631
ħc in MeV fm
- const double `gfermi_gev` = 1.16637e-5
Fermi coupling constant (G_F) in GeV⁻².
- const double `hc_mev_cm` = 1.973269631e-11
ħc in MeV cm

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 ħ = c = 1.
- const double `e2_mkssa` = `gsl_mkssa::electron_charge`
Electron charge squared in SI(MKSA) units.

7.9.1 Detailed Description

O2scl constants.

7.9.2 Variable Documentation

7.9.2.1 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 968 of file constants.h.

7.9.2.2 const double e2_hlorentz = gsl_num::fine_structure*4.0*pi

Electron charge squared in Heaviside-Lorentz units where ħ = 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 988 of file constants.h.

7.9.2.3 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 1014 of file constants.h.

7.10 o2scl_fm Namespace Reference

Constants in units of fm.

Variables

- const double [mev](#) = 1.0/o2scl_const::hc_mev_fm
1 MeV in fm⁻¹
- const double [kg](#) = mev/1.782661731e-30
1 kg in fm⁻¹
- const double [msun_per_km3](#) = gsl_mkسا::solar_mass/1.0e54*kg
1 M_☉/km³ in fm⁻⁴
- const double [Kelvin](#) = 8.617342e-11*mev
1 Kelvin in fm⁻¹
- const double [joule](#) = kg/gsl_mkسا::speed_of_light/gsl_mkسا::speed_of_light
1 Joule in fm⁻¹
- const double [erg](#) = kg/1.0e3/gsl_cgs::speed_of_light/gsl_cgs::speed_of_light
1 erg in fm⁻¹
- const double [sec](#) = gsl_mkسا::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⁻¹.
- const double [mass_muon](#) = 105.658357/o2scl_const::hc_mev_fm
Muon mass in fm⁻¹.
- const double [mass_amu](#) = 931.494013/o2scl_const::hc_mev_fm
Atomic mass unit in fm⁻¹.
- const double [mass_neutron](#) = 939.565/o2scl_const::hc_mev_fm
Neutron mass in fm⁻¹.
- const double [mass_proton](#) = 938.272/o2scl_const::hc_mev_fm
Proton mass in fm⁻¹.
- const double [mass_lambda](#) = 1115.683/o2scl_const::hc_mev_fm
Λ mass in fm⁻¹
- const double [mass_sigmam](#) = 1197.45/o2scl_const::hc_mev_fm
Σ⁻ mass in fm⁻¹
- const double [mass_sigma](#) = 1192.642/o2scl_const::hc_mev_fm
Σ⁰ mass in fm⁻¹
- const double [mass_sigmap](#) = 1189.37/o2scl_const::hc_mev_fm

- Σ^+ mass in fm⁻¹
• const double [mass_cascadem](#) = 1321.3/o2scl_const::hc_mev_fm
- Ξ^- mass in fm⁻¹
• const double [mass_cascade](#) = 1314.8/o2scl_const::hc_mev_fm
- Ξ^0 mass in fm⁻¹
• const double [mass_omega](#) = 782.57/o2scl_const::hc_mev_fm
- ω mass in fm⁻¹
• const double [mass_rho](#) = 769.3/o2scl_const::hc_mev_fm
- ρ mass in fm⁻¹

www.nist.gov

- const double [mass_alpha](#) = 3727.37905/o2scl_const::hc_mev_fm
Alpha particle mass in fm⁻¹.

7.10.1 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 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 fm⁻¹. Alternatively, [mev](#) is a number with units MeV⁻¹ · fm⁻¹. These can be combined, so that [erg](#) divided by [sec](#) is 1 erg/sec in units of fm⁻².

7.10.2 Variable Documentation

7.10.2.1 const double mass_alpha = 3727.37905/o2scl_const::hc_mev_fm

Alpha particle mass in fm⁻¹.

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

Definition at line 1079 of file constants.h.

7.11 o2scl_inte_qag_coeffs Namespace Reference

A namespace for the GSL adaptive integration coefficients.

Variables

- static const double [qk15_xgk](#) [8]
Abscissae of the 15-point Kronrod rule.
- static const double [qk15_wg](#) [4]
Weights of the 7-point Gauss rule.
- static const double [qk15_wgk](#) [8]
Weights of the 15-point Kronrod rule.
- static const double [qk21_xgk](#) [11]
Abscissae of the 21-point Kronrod rule.
- static const double [qk21_wg](#) [5]
Weights of the 10-point Gauss rule.
- static const double [qk21_wgk](#) [11]
Weights of the 21-point Kronrod rule.
- static const double [qk31_xgk](#) [16]
Abscissae of the 31-point Kronrod rule.
- static const double [qk31_wg](#) [8]
Weights of the 15-point Gauss rule.

- static const double [qk31_wgk](#) [16]
Weights of the 31-point Kronrod rule.
- static const double [qk41_xgk](#) [21]
Abscissae of the 41-point Kronrod rule.
- static const double [qk41_wg](#) [11]
Weights of the 20-point Gauss rule.
- static const double [qk41_wgk](#) [21]
Weights of the 41-point Kronrod rule.
- static const double [qk51_xgk](#) [26]
Abscissae of the 51-point Kronrod rule.
- static const double [qk51_wg](#) [13]
Weights of the 25-point Gauss rule.
- static const double [qk51_wgk](#) [26]
Weights of the 51-point Kronrod rule.
- static const double [qk61_xgk](#) [31]
Abscissae of the 61-point Kronrod rule.
- static const double [qk61_wg](#) [15]
Weights of the 30-point Gauss rule.
- static const double [qk61_wgk](#) [31]
Weights of the 61-point Kronrod rule.

7.11.1 Detailed Description

A namespace for the GSL adaptive integration coefficients.

In the abscissa arrays, the odd-indexed entries are the abscissae of the Gauss rule, while the even entries are the Kronrod abscissae which are added to the Gauss rule.

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.

7.12 o2scl_inte_qng_coeffs Namespace Reference

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

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]

7.12.1 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.

7.12.2 Variable Documentation

7.12.2.1 `const double w10[5]` [static]

Weights of the 10-point formula

Definition at line 51 of file `gsl_inte_qng.h`.

7.12.2.2 `const double w21a[5]` [static]

Weights of the 21-point formula for abscissae x_1

Definition at line 69 of file `gsl_inte_qng.h`.

7.12.2.3 `const double w21b[6]` [static]

Weights of the 21-point formula for abscissae x_2

Definition at line 78 of file `gsl_inte_qng.h`.

7.12.2.4 `const double w43a[10]` [static]

Weights of the 43-point formula for abscissae x_1, x_3

Definition at line 103 of file `gsl_inte_qng.h`.

7.12.2.5 `const double w43b[12]` [static]

Weights of the 43-point formula for abscissae x_3

Definition at line 117 of file `gsl_inte_qng.h`.

7.12.2.6 `const double w87a[21]` [static]

Weights of the 87-point formula for abscissae x_1, x_2, x_3

Definition at line 159 of file `gsl_inte_qng.h`.

7.12.2.7 `const double w87b[23]` [static]

Weights of the 87-point formula for abscissae x_4

Definition at line 184 of file `gsl_inte_qng.h`.

7.12.2.8 `const double x1[5]` [static]

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

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

7.12.2.9 `const double x2[5]` [static]

Abscissae common to the 21-, 43- and 87-point rule

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

7.12.2.10 `const double x3[11]` [static]

Abscissae common to the 43- and 87-point rule

Definition at line 88 of file gsl_inte_qng.h.

7.12.2.11 const double x4[22] [static]

Abscissae of the 87-point rule

Definition at line 133 of file gsl_inte_qng.h.

7.13 o2scl_linalg Namespace Reference

The namespace for linear algebra classes and functions.

Data Structures

- class [linear_solver](#)
A generic solver for the linear system $Ax = b$ [abstract base].
- class [linear_solver_lu](#)
Generic linear solver using LU decomposition.
- class [linear_solver_qr](#)
Generic linear solver using QR decomposition.
- class [linear_solver_hh](#)
Generic Householder linear solver.
- class [gsl_solver_LU](#)
GSL solver by LU decomposition.
- class [gsl_solver_QR](#)
GSL solver by QR decomposition.
- class [gsl_solver_HH](#)
GSL Householder solver.

Functions

- void [create_givens](#) (const double a, const double b, double &c, double &s)
Desc.
- template<class mat1_t, class mat2_t >
void [apply_givens_qr](#) (size_t M, size_t N, mat1_t &Q, mat2_t &R, size_t i, size_t j, double c, double s)
Apply a rotation to matrices from the QR decomposition.
- template<class mat1_t, class mat2_t >
void [apply_givens_lq](#) (size_t M, size_t N, mat1_t &Q, mat2_t &L, size_t i, size_t j, double c, double s)
Apply a rotation to matrices from the LQ decomposition.
- template<class vec_t >
void [apply_givens_vec](#) (vec_t &v, size_t i, size_t j, double c, double s)
Apply a rotation to a vector; $v \rightarrow G^T v$.
- template<class mat_t, class vec_t >
int [HH_solve](#) (size_t n, mat_t &A, const vec_t &b, vec_t &x)
Solve linear system after Householder decomposition.
- template<class mat_t, class vec_t >
int [HH_svx](#) (size_t N, size_t M, mat_t &A, vec_t &x)
Solve a linear system after Householder decomposition in place.
- template<class vec_t >
double [householder_transform](#) (const size_t n, vec_t &v)
Replace the vector v with a householder vector and a coefficient tau that annihilates the last $n-1$ elements of v .
- template<class mat_t >
double [householder_transform_subcol](#) (mat_t &A, const size_t ir, const size_t ic, const size_t n)
Compute the householder transform of a vector formed with the last n rows of a column of a matrix.

- template<class vec_t, class mat_t >
int [householder_hm](#) (const size_t M, const size_t N, double tau, const vec_t &v, mat_t &A)
Apply a householder transformation v, τ to matrix m .
- template<class mat_t >
int [householder_hm_sub](#) (mat_t &M, const size_t ir, const size_t ic, const size_t nr, const size_t nc, const mat_t &M2, const size_t ir2, const size_t ic2, double tau)
Apply a householder transformation v, τ to submatrix of m .
- template<class vec_t >
int [householder_hv](#) (const size_t N, double tau, const vec_t &v, vec_t &w)
Apply a householder transformation v to vector w .
- template<class mat_t, class vec_t >
int [householder_hv_sub](#) (const mat_t &M, vec_t &w, double tau, const size_t ie, const size_t N)
Apply a householder transformation v to vector w .
- template<class mat1_t, class mat2_t >
int [householder_hm_sub2](#) (const size_t M, const size_t ic, double tau, const mat1_t &mv, mat2_t &A)
Special version of householder transformation for [QR_unpack\(\)](#).
- template<class mat_t >
int [LU_decomp](#) (const size_t N, mat_t &A, o2scl::permutation &p, int &signum)
Compute the LU decomposition of the matrix A .
- template<class mat_t, class vec_t >
int [LU_solve](#) (const size_t N, const mat_t &LU, const o2scl::permutation &p, const vec_t &b, vec_t &x)
Solve a linear system after LU decomposition.
- template<class mat_t, class vec_t >
int [LU_svx](#) (const size_t N, const mat_t &LU, const o2scl::permutation &p, vec_t &x)
Solve a linear system after LU decomposition in place.
- template<class mat_t, class vec_t >
int [LU_refine](#) (const size_t N, const mat_t &A, const mat_t &LU, const o2scl::permutation &p, const vec_t &b, vec_t &x, vec_t &residual)
Refine the solution of a linear system.
- template<class mat_t, class mat_col_t >
int [LU_invert](#) (const size_t N, const mat_t &LU, const o2scl::permutation &p, mat_t &inverse)
Compute the inverse of a matrix from its LU decomposition.
- template<class mat_t >
double [LU_det](#) (const size_t N, const mat_t &LU, int signum)
Compute the determinant of a matrix from its LU decomposition.
- template<class mat_t >
double [LU_lndet](#) (const size_t N, const mat_t &LU)
Compute the logarithm of the absolute value of the determinant of a matrix from its LU decomposition.
- template<class mat_t >
int [LU_sgndet](#) (const size_t N, const mat_t &LU, int signum)
Compute the sign of the determinant of a matrix from its LU decomposition.
- template<class mat_t, class vec_t >
int [QR_decomp](#) (size_t M, size_t N, mat_t &A, vec_t &tau)
Compute the QR decomposition of matrix A .
- template<class mat_t, class vec_t, class vec2_t >
int [QR_solve](#) (size_t N, const mat_t &QR, const vec_t &tau, const vec2_t &b, vec2_t &x)
Solve the system $Ax = b$ using the QR factorization.
- template<class mat_t, class vec_t, class vec2_t >
int [QR_svx](#) (size_t M, size_t N, const mat_t &QR, const vec_t &tau, vec2_t &x)
Solve the system $Ax = b$ in place using the QR factorization.
- template<class mat_t, class vec_t, class vec2_t >
int [QR_QTvec](#) (const size_t M, const size_t N, const mat_t &QR, const vec_t &tau, vec2_t &v)
Form the product $Q^T v$ from a QR factorized matrix.
- template<class mat1_t, class mat2_t, class mat3_t, class vec_t >
int [QR_unpack](#) (const size_t M, const size_t N, const mat1_t &QR, const vec_t &tau, mat2_t &Q, mat3_t &R)
Unpack the QR matrix to the individual Q and R components.
- template<class mat1_t, class mat2_t, class vec1_t, class vec2_t >
int [QR_update](#) (size_t M, size_t N, mat1_t &Q, mat2_t &R, vec1_t &w, vec2_t &v)

Update a QR factorisation for $A = QR$, $A' = A + u v^T$.

- `template<class vec_t, class vec2_t>`
`int solve_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec_t &b, vec_t &x, size_t N)`
Solve a symmetric tridiagonal linear system.
- `template<class vec_t, class vec2_t>`
`int solve_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec2_t &belowdiag, const vec_t &rhs, vec_t &x, size_t N)`
Solve an asymmetric tridiagonal linear system.
- `template<class vec_t>`
`int solve_cyc_tridiag_sym (const vec_t &diag, const vec_t &offdiag, const vec_t &b, vec_t &x, size_t N)`
Solve a symmetric cyclic tridiagonal linear system.
- `template<class vec_t>`
`int solve_cyc_tridiag_nonsym (const vec_t &diag, const vec_t &abovediag, const vec_t &belowdiag, const vec_t &rhs, vec_t &x, size_t N)`
Solve an asymmetric cyclic tridiagonal linear system.

7.13.1 Detailed Description

The namespace for linear algebra classes and functions.

This namespace documentation is in the file [src/base/lib_settings.h](#)

7.13.2 Function Documentation

7.13.2.1 void o2scl_linalg::apply_givens_lq (size_t M, size_t N, mat1_t & Q, mat2_t & L, size_t i, size_t j, double c, double s) [inline]

Apply a rotation to matrices from the LQ decomposition.

This performs $Q \rightarrow QG$ and $L \rightarrow LG^T$.

Definition at line 83 of file [givens_base.h](#).

7.13.2.2 void o2scl_linalg::apply_givens_qr (size_t M, size_t N, mat1_t & Q, mat2_t & R, size_t i, size_t j, double c, double s) [inline]

Apply a rotation to matrices from the QR decomposition.

This performs $Q \rightarrow QG$ and $R \rightarrow G^T R$.

Definition at line 57 of file [givens_base.h](#).

7.13.2.3 int o2scl_linalg::householder_hm_sub (mat_t & M, const size_t ir, const size_t ic, const size_t nr, const size_t nc, const mat_t & M2, const size_t ir2, const size_t ic2, double tau) [inline]

Apply a householder transformation v , τ to submatrix of m .

Used in [QR_decomp\(\)](#).

Definition at line 163 of file [householder_base.h](#).

7.13.2.4 int o2scl_linalg::householder_hv_sub (const mat_t & M, vec_t & w, double tau, const size_t ie, const size_t N) [inline]

Apply a householder transformation v to vector w .

Used in [QR_QTvec\(\)](#).

Definition at line 231 of file [householder_base.h](#).

7.13.2.5 `double o2scl_linalg::householder_transform_subcol (mat_t & A, const size_t ir, const size_t ic, const size_t n)` `[inline]`

Compute the householder transform of a vector formed with the last n rows of a column of a matrix.

Used in [QR_decomp\(\)](#).

Definition at line 95 of file `householder_base.h`.

7.13.2.6 `int o2scl_linalg::LU_decomp (const size_t N, mat_t & A, o2scl::permutation & p, int & signum)` `[inline]`

Compute the LU decomposition of the matrix A .

On output the diagonal and upper triangular part of the input matrix A contain the matrix U . The lower triangular part of the input matrix (excluding the diagonal) contains L . The diagonal elements of L are unity, and are not stored.

The [permutation](#) matrix P is encoded in the [permutation](#) p . The j -th column of the matrix P is given by the k -th column of the identity matrix, where $k = p_j$ the j -th element of the [permutation](#) vector. The sign of the [permutation](#) is given by $signum$. It has the value $(-1)^n$, where n is the number of interchanges in the [permutation](#).

The algorithm used in the decomposition is Gaussian Elimination with partial pivoting (Golub & Van Loan, Matrix Computations, Algorithm 3.4.1).

Idea for future

The "swap rows j and i_{pivot} " section could probably be made more efficient using a "matrix_row"-like object.

Definition at line 73 of file `lu_base.h`.

7.13.2.7 `double o2scl_linalg::LU_det (const size_t N, const mat_t & LU, int signum)` `[inline]`

Compute the determinant of a matrix from its LU decomposition.

These functions compute the determinant of a matrix A from its LU decomposition, LU . The determinant is computed as the product of the diagonal elements of U and the sign of the row [permutation](#) $signum$.

Definition at line 252 of file `lu_base.h`.

7.13.2.8 `int o2scl_linalg::LU_invert (const size_t N, const mat_t & LU, const o2scl::permutation & p, mat_t & inverse)` `[inline]`

Compute the inverse of a matrix from its LU decomposition.

These functions compute the inverse of a matrix A from its LU decomposition (LU, p), storing the result in the matrix `inverse`. The inverse is computed by solving the system $Ax = b$ for each column of the identity matrix. It is preferable to avoid direct use of the inverse whenever possible, as the linear solver functions can obtain the same result more efficiently and reliably (consult any introductory textbook on numerical linear algebra for details).

Idea for future

could rewrite to avoid `mat_col_t`

Definition at line 217 of file `lu_base.h`.

7.13.2.9 `double o2scl_linalg::LU_Lndet (const size_t N, const mat_t & LU)` `[inline]`

Compute the logarithm of the absolute value of the determinant of a matrix from its LU decomposition.

These functions compute the logarithm of the absolute value of the determinant of a matrix A , $\ln |\det(A)|$, from its LU decomposition, LU . This function may be useful if the direct computation of the determinant would overflow or underflow.

Definition at line 275 of file `lu_base.h`.

7.13.2.10 `int o2scl_linalg::LU_refine (const size_t N, const mat_t & A, const mat_t & LU, const o2scl::permutation & p, const vec_t & b, vec_t & x, vec_t & residual)` [inline]

Refine the solution of a linear system.

These functions apply an iterative improvement to *x*, the solution of $A x = b$, using the LU decomposition of *A* into (LU,p). The initial residual $r = A x - b$ is also computed and stored in *residual*.

Definition at line 184 of file `lu_base.h`.

7.13.2.11 `int o2scl_linalg::LU_sgndet (const size_t N, const mat_t & LU, int signum)` [inline]

Compute the sign of the determinant of a matrix from its LU decomposition.

These functions compute the sign or phase factor of the determinant of a matrix *A*, $\det(A)/|\det(A)|$, from its LU decomposition, LU.

Definition at line 296 of file `lu_base.h`.

7.13.2.12 `int o2scl_linalg::LU_solve (const size_t N, const mat_t & LU, const o2scl::permutation & p, const vec_t & b, vec_t & x)` [inline]

Solve a linear system after LU decomposition.

This function solve the square system $A x = b$ using the LU decomposition of *A* into (LU, p) given by `gsl_linalg_LU_decomp` or `gsl_linalg_complex_LU_decomp`.

Definition at line 135 of file `lu_base.h`.

7.13.2.13 `int o2scl_linalg::LU_svx (const size_t N, const mat_t & LU, const o2scl::permutation & p, vec_t & x)` [inline]

Solve a linear system after LU decomposition in place.

These functions solve the square system $A x = b$ in-place using the LU decomposition of *A* into (LU,p). On input *x* should contain the right-hand side *b*, which is replaced by the solution on output.

Definition at line 156 of file `lu_base.h`.

7.13.2.14 `int o2scl_linalg::QR_update (size_t M, size_t N, mat1_t & Q, mat2_t & R, vec1_t & w, vec2_t & v)` [inline]

Update a QR factorisation for $A = Q R$, $A' = A + u v^T$.

M and *N* are the number of rows and columns of *R*.

```
* Q' R' = QR + u v^T
*       = Q (R + Q^T u v^T)
*       = Q (R + w v^T)
*
* where w = Q^T u.
*
* Algorithm from Golub and Van Loan, "Matrix Computations", Section
* 12.5 (Updating Matrix Factorizations, Rank-One Changes)
```

Definition at line 166 of file `qr_base.h`.

7.13.2.15 `int o2scl_linalg::solve_cyc_tridiag_nonsym (const vec_t & diag, const vec_t & abovediag, const vec_t & belowdiag, const vec_t & rhs, vec_t & x, size_t N)` [inline]

Solve an asymmetric cyclic tridiagonal linear system.

This function solves the following system w/o the corner elements and then use Sherman-Morrison formula to compensate for them

```
*
*      diag[0]  abovediag[0]          0      ..... belowdiag[N-1]
*  belowdiag[0]      diag[1]  abovediag[1]  .....
*          0  belowdiag[1]      diag[2]
*          0          0  belowdiag[2]  .....
*          ...          ...
*  abovediag[N-1]      ...
```

Definition at line 319 of file tridiag_base.h.

7.13.2.16 `int o2scl_linalg::solve_cyc_tridiag_sym (const vec_t & diag, const vec_t & offdiag, const vec_t & b, vec_t & x, size_t N)` [inline]

Solve a symmetric cyclic tridiagonal linear system.

Todo

Put reference in correctly.

For a description of the method see [Engeln-Mullges + Uhlig, p. 96]

```
*
*      diag[0]  offdiag[0]          0      ..... offdiag[N-1]
*  offdiag[0]      diag[1]  offdiag[1]  .....
*          0  offdiag[1]      diag[2]
*          0          0  offdiag[2]  .....
*          ...          ...
*  offdiag[N-1]      ...
```

Definition at line 216 of file tridiag_base.h.

7.13.2.17 `int o2scl_linalg::solve_tridiag_nonsym (const vec_t & diag, const vec2_t & abovediag, const vec2_t & belowdiag, const vec_t & rhs, vec_t & x, size_t N)` [inline]

Solve an asymmetric tridiagonal linear system.

This function uses plain gauss elimination, only not bothering with the zeroes.

```
*
*      diag[0]  abovediag[0]          0      .....
*  belowdiag[0]      diag[1]  abovediag[1]  .....
*          0  belowdiag[1]      diag[2]
*          0          0  belowdiag[2]  .....
```

Definition at line 145 of file tridiag_base.h.

7.13.2.18 `int o2scl_linalg::solve_tridiag_sym (const vec_t & diag, const vec2_t & offdiag, const vec_t & b, vec_t & x, size_t N)` [inline]

Solve a symmetric tridiagonal linear system.

Idea for future

Convert into class for memory managment and combine with other functions below

For description of method see [Engeln-Mullges + Uhlig, p. 92]

```
*
*      diag[0]   offdiag[0]           0   .....
*  offdiag[0]      diag[1]   offdiag[1]   .....
*           0   offdiag[1]      diag[2]
*           0           0   offdiag[2]   .....
```

Definition at line 68 of file tridiag_base.h.

7.14 o2scl_linalg_paren Namespace Reference

The namespace for linear algebra classes and functions with operator().

7.14.1 Detailed Description

The namespace for linear algebra classes and functions with operator().

This namespace contains an identical copy of all the functions given in the [o2scl_cblas](#) namespace, but perform array indexing with `operator()` rather than `operator[]`. See [o2scl_linalg](#) for the function listing and documentation.

This namespace documentation is in the file [src/base/lib_settings.h](#)

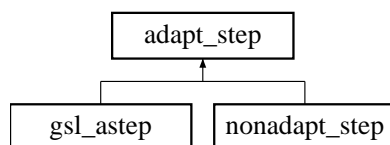
8 Data Structure Documentation

8.1 adapt_step Class Template Reference

Adaptive stepper [abstract base].

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

Inheritance diagram for `adapt_step`:



8.1.1 Detailed Description

```
template<class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_base, 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 [abstract base].

The adaptive stepper routines are based on several applications of ordinary ODE steppers (implemented in [odestep](#)). Each adaptive stepper ([gsl_astep](#) or [nonadapt_step](#)) can be used with any of the ODE stepper classes (e.g. [gsl_rkck](#)). By default, [gsl_rkck](#) is used. To modify the ODE stepper which is used, use the member function [set_step\(\)](#) documented below.

Note:

If you use [gsl_rkck_fast](#) or [gsl_rk8pd_fast](#), you'll need to make sure that the argument `n` to [astep\(\)](#) or [astep_derivs\(\)](#) below matches the template size parameter given in the ODE stepper.

Definition at line 52 of file adapt_step.h.

Public Member Functions

- virtual int [astep](#) (double &x, double &h, double xmax, size_t n, vec_t &y, vec_t &dydx_out, vec_t &yerr, param_t &pa, func_t &derivs)=0
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, vec_t &yerr, param_t &pa, func_t &derivs)=0
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.

8.1.2 Member Function Documentation

8.1.2.1 virtual int astep (double & x, double & h, double xmax, size_t n, vec_t & y, vec_t & dydx_out, vec_t & yerr, param_t & pa, func_t & derivs) [pure 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, `dydx_out` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

Implemented in [gsl_astep](#), and [nonadapt_step](#).

8.1.2.2 virtual int astep_derivs (double & x, double & h, double xmax, size_t n, vec_t & y, vec_t & dydx, vec_t & yerr, param_t & pa, func_t & derivs) [pure 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, `dydx` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

Implemented in [gsl_astep](#), and [nonadapt_step](#).

8.1.2.3 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 106 of file adapt_step.h.

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

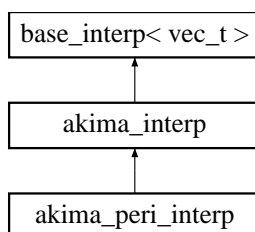
- `adapt_step.h`

8.2 akima_interp Class Template Reference

Akima spline interpolation (GSL).

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

Inheritance diagram for `akima_interp`::



8.2.1 Detailed Description

```
template<class vec_t> class akima_interp< vec_t >
```

Akima spline interpolation (GSL).

Idea for future

It appears that the `interp()` function below searches for indices slightly differently than the original GSL `eval()` function. This should be checked, as it might be slightly non-optimal in terms of speed (shouldn't matter for the accuracy).

Definition at line 729 of file `interp.h`.

Public Member Functions

- `akima_interp` (bool periodic=false)
Create a base interpolation object with or without periodic boundary conditions.
- virtual int `allocate` (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 aa, double bb, double &result)
Give the value of the integral $\int_a^b y(x) dx$.
- virtual const char * `type` ()
Return the type, "akima_interp".

Protected Member Functions

- void [akima_calc](#) (const vec_t &x_array, size_t size, double m[])

For initializing the interpolation.

Protected Attributes

- bool [peri](#)

True for periodic boundary conditions.

Storage for Akima spline interpolation

- double * **b**
- double * **c**
- double * **d**
- double * **um**

Private Member Functions

- [akima_interp](#) (const [akima_interp](#)< vec_t > &)
- [akima_interp](#)< vec_t > & **operator=** (const [akima_interp](#)< vec_t > &)

8.2.2 Member Function Documentation

8.2.2.1 virtual int init (const vec_t & xa, const vec_t & ya, size_t size) [inline, virtual]

Initialize interpolation routine.

Periodic boundary conditions

Non-periodic boundary conditions

Reimplemented from [base_interp](#).

Definition at line 832 of file interp.h.

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

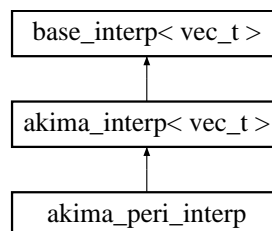
- interp.h

8.3 akima_peri_interp Class Template Reference

Akima spline interpolation with periodic boundary conditions (GSL).

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

Inheritance diagram for akima_peri_interp::



8.3.1 Detailed Description

template<class vec_t> class akima_peri_interp< vec_t >

Akima spline interpolation with periodic boundary conditions (GSL).

This is convenient to allow interpolation objects to be supplied as template parameters

Definition at line 1005 of file interp.h.

Public Member Functions

- virtual const char * [type](#) ()
Return the type, "akima_peri_interp".

Private Member Functions

- **akima_peri_interp** (const [akima_peri_interp](#)< vec_t > &)
- [akima_peri_interp](#)< vec_t > & **operator=** (const [akima_peri_interp](#)< vec_t > &)

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

- interp.h

8.4 anneal_mt Class Template Reference

Multidimensional minimization by simulated annealing (Boost multi-threaded version).

```
#include <anneal_mt.h>
```

8.4.1 Detailed Description

template<class param_t, class param_vec_t = std::vector<param_t>, class func_t = multi_funct<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class rng_t = gsl_rnga> class anneal_mt< param_t, param_vec_t, func_t, vec_t, alloc_vec_t, alloc_t, rng_t >

Multidimensional minimization by simulated annealing (Boost multi-threaded version).

See also the unthreaded version of this class in [gsl_anneal](#).

Idea for future

There may be a good way to remove the function indirection here to make this class a bit faster.

Definition at line 51 of file anneal_mt.h.

Public Member Functions

- virtual const char * [type](#) ()
Return string denoting type ("anneal_mt").
- virtual int [print_iter](#) (size_t nv, vec_t &xx, double y, int iter, double tptr, std::string comment)
Print out iteration information.

Basic usage

- virtual int **mmin** (size_t nv, vec_t &x0, double &fmin, func_t &func, size_t np, param_vec_t &pars)
Calculate the minimum fmin of func w.r.t the array x0 of size nv using np threads.

Iteration control

- virtual int **next** (size_t nv, vec_t &x_old, double min_old, vec_t &x_new, double min_new, double &T, size_t n_moves, bool &finished)
Determine how to change the minimization for the next iteration.
- virtual int **start** (size_t nv, double &T)
Setup initial temperature and stepsize.

Data Fields

- rng_t **def_rng**
The default random number generator.

Parameters

- double **boltz**
Boltzmann factor (default 1.0).
- int **ntrial**
Number of iterations.
- int **verbose**
Output control.
- double **tolx**
The independent variable tolerance.

Protected Member Functions

- void **func_wrapper** (size_t ip)
The function wrapper executed by thread with index ip.
- virtual int **allocate** (size_t nv, size_t np)
Allocate memory for a minimizer over n dimensions with stepsize step.
- virtual int **free** (size_t np)
Free allocated memory.
- virtual int **step** (vec_t &sx, int nv)
Make a step to a new attempted minimum.

Protected Attributes

- std::ostream * **outs**
Stream for verbose output.
- std::istream * **ins**
Stream for verbose input.
- size_t **nproc**
The number of threads to run.
- param_vec_t * **params**
The user-specified parameters.
- size_t **nvar**
The number of variables over which we minimize.
- func_t * **f**
The function to minimize.
- alloc_t **ao**
Allocation object.
- size_t **nstep**
Number of step sizes.
- double * **step_sizes**

Step sizes.

Storage for present, next, and best vectors

- `alloc_vec_t x`
- `alloc_vec_t * new_x`
- `alloc_vec_t best_x`
- `alloc_vec_t new_E`
- `alloc_vec_t old_x`

8.4.2 Member Function Documentation

8.4.2.1 `virtual int print_iter (size_t nv, vec_t & xx, 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 247 of file `anneal_mt.h`.

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

- `anneal_mt.h`

8.5 array_2d_alloc Class Template Reference

A simple class to provide an `allocate()` function for 2-dimensional arrays.

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

8.5.1 Detailed Description

```
template<class mat_t> class array_2d_alloc< mat_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 an analog to `pointer_2d_alloc` and `omatrix_alloc`. This class is used in `gsl_mroot_hybrids_ts.cpp`.

Definition at line 119 of file `array.h`.

Public Member Functions

- void `allocate` (`mat_t &v`, `size_t i`, `size_t j`)
Allocate v for i elements.
- void `free` (`mat_t &v`, `size_t i`)
Free memory.

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

- `array.h`

8.6 array_2d_col Class Template Reference

Column of a 2d array.

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

8.6.1 Detailed Description

```
template<size_t R, size_t C, class data_t = double> class array_2d_col< R, C, data_t >
```

Column of a 2d array.

This works because two-dimensional arrays are always contiguous (as indicated in appendix C of Soustroup's book).

Definition at line 289 of file array.h.

Public Member Functions

- [array_2d_col](#) (data_t mat[R][C], size_t i)
Create an object as the i-th column of mat.
- data_t & [operator\[\]](#) (size_t i)
Array-like indexing.
- const data_t & [operator\[\]](#) (size_t i) const
Array-like indexing.

Protected Attributes

- data_t * a
The array pointer.

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

- [array.h](#)

8.7 array_2d_row Class Template Reference

Row of a 2d array.

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

8.7.1 Detailed Description

```
template<class array_2d_t, class data_t = double> class array_2d_row< array_2d_t, data_t >
```

Row of a 2d array.

Definition at line 327 of file array.h.

Public Member Functions

- [array_2d_row](#) (array_2d_t &mat, size_t i)
Create an object as the i-th row of mat.
 - data_t & [operator\[\]](#) (size_t i)
Array-like indexing, returns the element in the i-th column of the chosen row.
-

- `const data_t & operator[] (size_t i) const`
Array-like indexing, returns the element in the `i`th column of the chosen row.

Protected Attributes

- `data_t * a`
The array pointer.

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

- [array.h](#)

8.8 array_alloc Class Template Reference

A simple class to provide an `allocate()` function for arrays.

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

8.8.1 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 an analog to `pointer_alloc` and `ovector_alloc`. This class is used, for example, for `sma_interp_vec`.

Idea for future

Might it be possible to rework this so that it does range checking and ensures that the user doesn't try to allocate more or less space? I.e. `array_alloc<double[2]>` complains if you try an `allocate(x,3)`?

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

Public Member Functions

- `void allocate (vec_t &v, size_t 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](#)

8.9 array_const_reverse Class Template Reference

A simple class which reverses the order of an array.

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

8.9.1 Detailed Description

`template<size_t sz> class array_const_reverse< sz >`

A simple class which reverses the order of an array.

See an example of usage in `interp_ts.cpp` and `smart_interp_ts.cpp`.

Definition at line 211 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`

8.10 array_const_subvector Class Reference

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

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

8.10.1 Detailed Description

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

Idea for future

Make the member data truly const and remove the extra typecast.

Definition at line 369 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](#)

8.11 array_const_subvector_reverse Class Reference

Reverse a subvector of a const array.

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

8.11.1 Detailed Description

Reverse a subvector of a const array.

Definition at line 452 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:

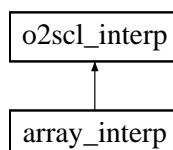
- [array.h](#)

8.12 array_interp Class Template Reference

A specialization of [o2scl_interp](#) for C-style double arrays.

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

Inheritance diagram for array_interp::



8.12.1 Detailed Description

template<size_t n> class array_interp< n >

A specialization of [o2scl_interp](#) for C-style double arrays.

Definition at line 1560 of file interp.h.

Public Member Functions

- [array_interp](#) ([base_interp_mgr](#)< double[n]> &it, [base_interp_mgr](#)< [array_const_reverse](#)< n > > &rit)
Create with base interpolation objects it and rit.
- [array_interp](#) ()
Create an interpolator using the default base interpolation objects.

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

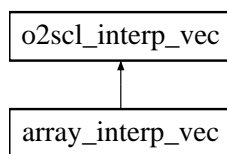
- interp.h

8.13 array_interp_vec Class Template Reference

A specialization of [o2scl_interp_vec](#) for C-style arrays.

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

Inheritance diagram for array_interp_vec::



8.13.1 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 1589 of file interp.h.

Public Member Functions

- [array_interp_vec](#) ([base_interp_mgr](#)< 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

8.14 array_reverse Class Template Reference

A simple class which reverses the order of an array.

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

8.14.1 Detailed Description

template<size_t sz> class array_reverse< sz >

A simple class which reverses the order of an array.

See an example of usage in `interp_ts.cpp` and `smart_interp_ts.cpp`.

Definition at line 171 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](#)

8.15 `array_subvector` Class Reference

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

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

8.15.1 Detailed Description

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

Definition at line 241 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](#)

8.16 array_subvector_reverse Class Reference

Reverse a subvector of an array.

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

8.16.1 Detailed Description

Reverse a subvector of an array.

Definition at line 407 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:

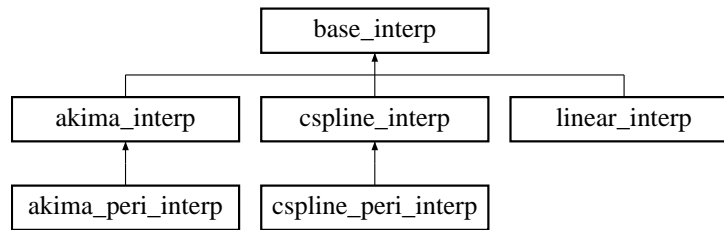
- [array.h](#)

8.17 base_interp Class Template Reference

Base low-level interpolation class [abstract base].

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

Inheritance diagram for base_interp::



8.17.1 Detailed Description

template<class vec_t> class base_interp< vec_t >

Base low-level interpolation class [abstract base].

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 [allocate\(\)](#) 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. (I think I might have removed the acceleration)

Definition at line 67 of file interp.h.

Public Member Functions

- virtual int [allocate](#) (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)=0
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)=0
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)=0
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)=0
Give the value of the integral $\int_a^b y(x) dx$.
- virtual const char * [type](#) ()
Return the type, "base_interp".

Data Fields

- size_t [min_size](#)
The minimum size of the vectors to interpolate between.

Protected Member Functions

- 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 for both the cspline and Akima types.

Protected Attributes

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

Private Member Functions

- `base_interp` (const `base_interp< vec_t > &`)
- `base_interp< vec_t > & operator=` (const `base_interp< vec_t > &`)

8.17.2 Field Documentation

8.17.2.1 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 108 of file `interp.h`.

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

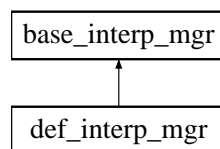
- `interp.h`

8.18 base_interp_mgr Class Template Reference

A base interpolation object manager [abstract base].

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

Inheritance diagram for `base_interp_mgr`:



8.18.1 Detailed Description

```
template<class vec_t> class base_interp_mgr< vec_t >
```

A base interpolation object manager [abstract base].

Definition at line 1034 of file `interp.h`.

Public Member Functions

- virtual `base_interp< vec_t > * new_interp` ()=0

Create a new interpolation object.

- virtual int [free_interp](#) ([base_interp](#)< vec_t > *b)
Deallocate an interpolation object.

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

- interp.h

8.19 base_ioc Class Reference

Setup I/O objects for base library classes.

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

8.19.1 Detailed Description

Setup I/O objects for base library classes.

This class is experimental.

Definition at line 42 of file base_ioc.h.

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_units_io_type](#) * [table_units_io](#)
- [table_io_type](#) * [table_io](#)

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

- base_ioc.h

8.20 bin_size Class Reference

Determine bin size (CERNLIB).

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

8.20.1 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

This class is not working yet.

Definition at line 47 of file `bin_size.h`.

Public Member Functions

- `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)

8.20.2 Member Function Documentation

8.20.2.1 `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 ($BL \leq AL$)
- `bh` - Upper limit ($BH \geq AH$)
- `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:

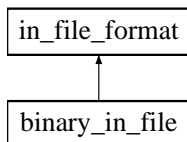
- `bin_size.h`

8.21 `binary_in_file` Class Reference

Binary input file.

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

Inheritance diagram for `binary_in_file`::



8.21.1 Detailed Description

Binary input file.

This class is experimental.

Definition at line 139 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:

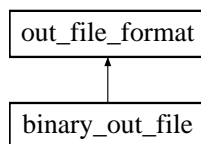
- `binary_file.h`

8.22 binary_out_file Class Reference

Binary output file.

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

Inheritance diagram for binary_out_file::



8.22.1 Detailed Description

Binary output file.

This class is experimental.

Definition at line 42 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:

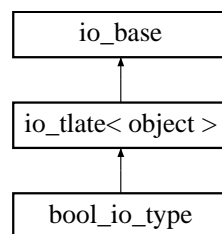
- [binary_file.h](#)

8.23 bool_io_type Class Reference

I/O object for bool variables.

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

Inheritance diagram for bool_io_type::



8.23.1 Detailed Description

I/O object for bool variables.

This class is experimental.

Definition at line 2238 of file collection.h.

Public Member Functions

- [bool_io_type](#) (const char *t)
Desc.

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

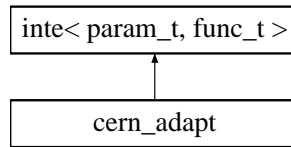
- [collection.h](#)

8.24 cern_adapt Class Template Reference

Adaptive integration (CERNLIB).

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

Inheritance diagram for cern_adapt::



8.24.1 Detailed Description

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

Adaptive integration (CERNLIB).

Uses a base integration object (default is [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 half. 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\(\)](#).

This class is based on the CERNLIB routines RADAPT and DADAPT which are documented at <http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d102/top.html>

Idea for future

- Allow user to set the initial segments?
- It might be interesting to directly compare the performance of this class to [gsl_inte_qag](#).

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

Public Member Functions

- `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 &errs)`
*Return the *i*th segment.*
- `template<class vec_t >`
`int get_segments (vec_t &xlow, vec_t &xhigh, vec_t &value, vec_t &errs)`
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.

8.24.2 Field Documentation

8.24.2.1 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:

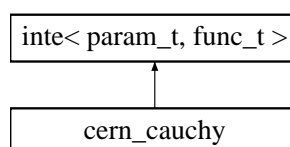
- `cern_adapt.h`

8.25 cern_cauchy Class Template Reference

Cauchy principal value integration (CERNLIB).

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

Inheritance diagram for `cern_cauchy`:



8.25.1 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()`. This is different from how the `gsl_inte_qawc` operates.

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_einval` - Singularity is on an endpoint
- `gsl_efailed` - Couldn't reach requested accuracy (occurs only if `inte::err_nonconv` is true)

This function is based on the CERNLIB routines `RCAUCH` and `DCAUCH` which are documented at <http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d104/top.html>

Definition at line 64 of file `cern_cauchy.h`.

Public Member Functions

- `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:

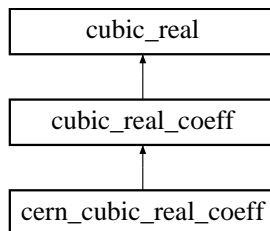
- `cern_cauchy.h`

8.26 cern_cubic_real_coeff Class Reference

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

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

Inheritance diagram for cern_cubic_real_coeff:



8.26.1 Detailed Description

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

This follows the original Fortran code except in the function `rrteq3()`, the roots are returned in `x[0]`, `x[1]`, and `x[2]` instead of `x[1]`, `x[2]`, and `x[3]`. (The arrays `y` and `z` were already zero-indexed in the CERN routine.) Similar to the original, in the case of complex roots, `x[0]` is the real `root` and `x[1]` and `x[2]` contain the real and imaginary parts of the complex roots.

Another small change is that the discriminant for the resolvent cubic is evaluated slightly differently in order to improve the properties in the case where the roots are not of order unity. The default CERNLIB behavior can be restored by setting `improve_scale` to `false`.

Definition at line 403 of file `poly.h`.

Public Member Functions

- 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").

Data Fields

- double `eps`
Numerical tolerance (default 10^{-6}).
- double `delta`
Numerical tolerance (default 10^{-15}).
- bool `improve_scale`
Improve algorithm for bad-scaled roots (default true).

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

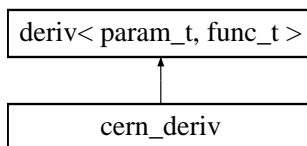
- `poly.h`

8.27 cern_deriv Class Template Reference

Numerical differentiation routine (CERNLIB).

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

Inheritance diagram for cern_deriv::



8.27.1 Detailed Description

```
template<class param_t, class func_t = funct<param_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.

For sufficiently difficult functions, the derivative computation can fail, and will call the error handler and return zero with zero error.

Based on the CERNLIB routine DERIV, which was based on [Rutishauser63](#) and is documented at <http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d401/top.html>

An example demonstrating the usage of this class is given in `examples/ex_deriv.cpp` and the [Numerical differentiation example](#).

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 69 of file `cern_deriv.h`.

Public Member Functions

- 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 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 5×10^{14}).

Protected Member Functions

- virtual int `calc_err_int` (double `x`, typename `deriv`< `param_t`, `func_t` >::`dpars` &`pa`, `func_t`< typename `deriv`< `param_t`, `func_t` >::`dpars` > &`func`, double &`dfdx`, double &`err`)
Calculate the first derivative of `func` w.r.t. `x`.

Protected Attributes

Storage for the fixed coefficients

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

8.27.2 Member Function Documentation

8.27.2.1 virtual int `calc_err_int` (double `x`, typename `deriv`< `param_t`, `func_t` >::`dpars` &`pa`, `func_t`< typename `deriv`< `param_t`, `func_t` >::`dpars` > &`func`, double &`dfdx`, double &`err`) [`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 222 of file `cern_deriv.h`.

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

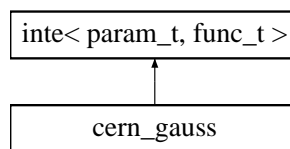
- `cern_deriv.h`

8.28 cern_gauss Class Template Reference

Gaussian quadrature (CERNLIB).

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

Inheritance diagram for `cern_gauss`:



8.28.1 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 λ 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.

If the desired accuracy is not achieved, the integration functions will call the error handler and return the best guess, unless `inte::err_nonconv` is false, in which case the error handler is not called.

This function is based on the CERNLIB routines GAUSS and DGAUSS which are documented at <http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d103/top.html>

Idea for future

Allow user to change `cst`?

Definition at line 94 of file `cern_gauss.h`.

Public Member Functions

- 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]

8.28.2 Member Function Documentation

8.28.2.1 virtual double integ (func_t &func, double a, double b, param_t &pa) [inline, virtual]

Integrate function `func` from `a` to `b`.

Idea for future

Modify this to include iteration count information as done in [cern_cauchy](#)

Implements [inte](#).

Definition at line 144 of file `cern_gauss.h`.

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

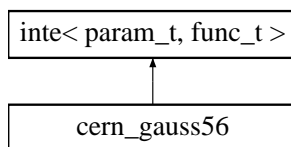
- `cern_gauss.h`

8.29 cern_gauss56 Class Template Reference

5,6-point Gaussian quadrature (CERNLIB)

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

Inheritance diagram for `cern_gauss56`:



8.29.1 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 $|I_5 - I_6|$.

This class is based on the CERNLIB routines RGS56P and DGS56P which are documented at <http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d106/top.html>

Definition at line 45 of file `cern_gauss56.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 &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]

8.29.2 Member Function Documentation

8.29.2.1 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` giving result `res` and error `err`.

This function always returns [gsl_success](#).

Implements [inte](#).

Definition at line 90 of file `cern_gauss56.h`.

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

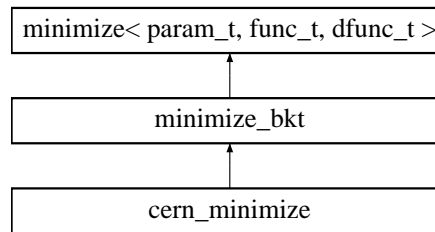
- `cern_gauss56.h`

8.30 cern_minimize Class Template Reference

One-dimensional minimization (CERNLIB).

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

Inheritance diagram for `cern_minimize`:



8.30.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class cern_minimize< param_t, func_t >

One-dimensional minimization (CERNLIB).

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 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 the CERNLIB routines RMINFC and DMINFC, which was based on [Fletcher87](#), and [Krabs83](#) and is documented at <http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d503/top.html>

Definition at line 61 of file cern_minimize.h.

Public Member Functions

- 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 δ .
- virtual const char * [type](#) ()
Return string denoting type ("cern_minimize").

Data Fields

- int [min_type](#)
Type of minimum found.

Protected Member Functions

- int [nint](#) (double x)
C analog of Fortran's "Nearest integer" function.

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.

8.30.2 Member Function Documentation

8.30.2.1 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.

Implements [minimize_bkt](#).

Definition at line 99 of file cern_minimize.h.

8.30.2.2 int [set_delta](#) (double d) [inline]

Set the value of δ .

If this is not called before [min_bkt\(\)](#) is used, then the suggested value $\delta = 10\text{tol}x$ is used.

Definition at line 176 of file cern_minimize.h.

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

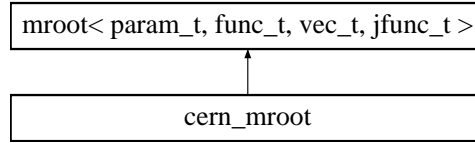
- cern_minimize.h

8.31 cern_mroot Class Template Reference

Multi-dimensional mroot-finding routine (CERNLIB).

```
#include <cern_mroot.h>
```

Inheritance diagram for cern_mroot::



8.31.1 Detailed Description

```
template<class param_t, class func_t = mm_func<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc, class jfunc_t = jac_func<param_t,vec_t,omatrix_base>> 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

$$\begin{aligned}
 1 : \quad & \max |f_i(x)| \leq \text{tol}f \\
 2 : \quad & \max |x_i - x'_i| \leq \text{tol}x \times \max |x_i|
 \end{aligned}$$

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.

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.

There is an example for the usage of the multidimensional solver classes given in `examples/ex_mroot.cpp`, see [Multidimensional solver example](#).

Idea for future

Modify this so it handles functions which return non-zero values.

Idea for future

Move some of the memory allocation out of `msolve()`

Idea for future

Give the user access to the number of function calls

Idea for future

Rename nier6, nier7, and nier8 to something sensible.

Based on the CERNLIB routines RSNLEQ and DSNLEQ, which was based on [More79](#) and [More80](#) and is documented at <http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/c201/top.html>

Definition at line 89 of file cern_mroot.h.

Public Member Functions

- `int get_info ()`
Get the value of INFO from the last call to `msolve()`.
- `std::string get_info_string ()`
Get the a string corresponding to the integer returned by `cern_mroot::get_info()`.
- `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.

8.31.2 Member Function Documentation

8.31.2.1 `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 ≤ 0 .

The return values returned by `msolve()` corresponding to the values of INFO above are 1 - `gsl_success` 2 - `gsl_success` 3 - `gsl_success` 4 - `gsl_emaxiter` 5 - `gsl_esing` 6 - `gsl_enoprogram` 7 - `gsl_erunaway` 8 - `gsl_efailed` 9 - `gsl_einval`

Definition at line 160 of file `cern_mroot.h`.

8.31.3 Field Documentation

8.31.3.1 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 226 of file `cern_mroot.h`.

8.31.3.2 int maxf

Maximum number of function evaluations.

If `maxf` ≤ 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 197 of file `cern_mroot.h`.

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

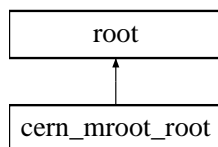
- `cern_mroot.h`

8.32 cern_mroot_root Class Template Reference

One-dimensional version of `cern_mroot`.

```
#include <cern_mroot_root.h>
```

Inheritance diagram for `cern_mroot_root::`



8.32.1 Detailed Description

template<class param_t = int, 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

$$\begin{aligned} 1 : \quad & \max |f_i(x)| \leq \text{tolf} \\ 2 : \quad & \max |x_i - x'_i| \leq \text{tolx} \times \max |x_i| \end{aligned}$$

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 64 of file `cern_mroot_root.h`.

Public Member Functions

- 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.

8.32.2 Member Function Documentation

8.32.2.1 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 102 of file `cern_mroot_root.h`.

8.32.3 Field Documentation

8.32.3.1 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
```

Idea for future

This number should probably default to one of the GSL tolerances.

Definition at line 143 of file `cern_mroot_root.h`.

8.32.3.2 int maxf

Maximum number of function evaluations.

If `maxf` ≤ 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 111 of file `cern_mroot_root.h`.

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

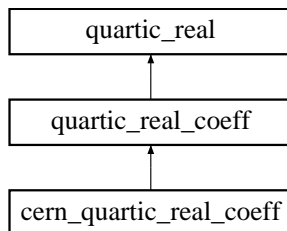
- `cern_mroot_root.h`

8.33 cern_quartic_real_coeff Class Reference

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

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

Inheritance diagram for cern_quartic_real_coeff::



8.33.1 Detailed Description

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

Definition at line 444 of file poly.h.

Public Member Functions

- 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 $x = x_1$, $x = x_2$, $x = x_3$, and $x = x_4$.
- 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.

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

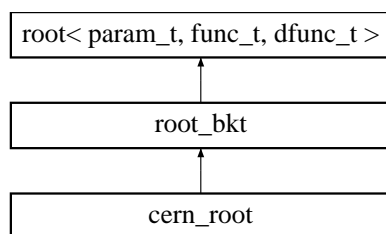
- [poly.h](#)

8.34 cern_root Class Template Reference

One-dimensional root-finding routine (CERNLIB).

```
#include <cern_root.h>
```

Inheritance diagram for cern_root::



8.34.1 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_1 and x_2 in $[a, b]$ such that $f(x_1)f(x_2) \leq 0$, $|f(x_1)| \leq |f(x_2)|$, and $|x_1 - x_2| \leq 2 \text{tolx} (1 + |x_0|)$. The function `solve_bkt()` requires inputs `x1` and `x2` such that $f(x_1)f(x_2) \leq 0$.

The variable `cern_root::tolx` defaults to 10^{-8} and `cern_root::ntrial` defaults to 200.

The function `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 the CERNLIB routines RZEROX and DZEROX, which was based on [Bus75](http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/c200/top.html) and is documented at <http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/c200/top.html>

Definition at line 61 of file `cern_root.h`.

Public Member Functions

- 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.

8.34.2 Member Function Documentation

8.34.2.1 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 112 of file cern_root.h.

8.34.2.2 virtual int solve_bkt(double &x1, double x2, param_t &pa, func_t &func) [inline, virtual]

Solve func in region $x_1 < x < x_2$ returning x_1 .

The parameters x1 and x2 should be set so that $f(x_1)f(x_2) \leq 0$ before calling solve_bkt(). If this is not the case, the error handler will be called and the solver will fail.

This function converges unless the number of iterations is larger than root::ntrial, in which case root::last_conv is set to gsl_emaxiter and the error handler is called if root::err_nonconv is true.

Implements root_bkt.

Definition at line 138 of file cern_root.h.

8.34.3 Field Documentation

8.34.3.1 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 80 of file cern_root.h.

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

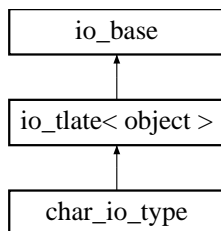
- cern_root.h

8.35 char_io_type Class Reference

I/O object for char variables.

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

Inheritance diagram for char_io_type::



8.35.1 Detailed Description

I/O object for char variables.

This class is experimental.

Definition at line 2264 of file collection.h.

Public Member Functions

- [char_io_type](#) (const char *t)
Desc.

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

- [collection.h](#)

8.36 cinput Class Reference

Class to control object input.

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

8.36.1 Detailed Description

Class to control object input.

This class is experimental.

Definition at line 1414 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:

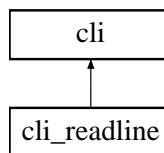
- [collection.h](#)

8.37 cli Class Reference

Configurable command-line interface.

```
#include <cli.h>
```

Inheritance diagram for cli::



8.37.1 Detailed Description

Configurable command-line interface.

This class is experimental.

Default commands: help, get/set, quit, exit, '!', verbose, license, warranty, alias, run.

Note that if the shell command is allowed (as it is by default) there are some potential security issues which are not solved here.

Commands which begin with a '#' character are ignored.

Note:

In interactive mode, commands are limited to 300 characters.

Todo

There are some fixme entries in cli.cpp associated with when cop is zero

Todo

Only add get() and set() commands if some parameters are set

Todo

Warn in [run_interactive\(\)](#) when extra parameters are given

Idea for future

Include a "remove command" function

Idea for future

A replace command function, there's already some code in cli.cpp for this.

Idea for future

There's some code duplication between `comm_option_run()` and `run_interactive()`

Idea for future

Allow the user to set the tilde string

Concepts

As a matter of definition, the command-line arguments are simply called arguments. These arguments may be options (in which case they begin with either one dash or two) or parameters to these options. When run in interactive mode, these options are also commands.

Definition at line 193 of file `cli.h`.

Public Member Functions

- int `set_function` (`comm_option_func` &usf)
Function to call when a set command is issued.
- virtual char * `cli_gets` (const char *c)
The function which obtains input from the user.
- 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 from a const char array.
- int `process_args` (std::string s, std::vector< `cmd_line_arg` > &ca, int debug=0)
Process command-line arguments from a string.
- int `set_verbose` (int v)
Set verbosity.
- int `run_interactive` ()
Run the interactive mode.
- int `set_alias` (std::string alias, std::string str)
Set an alias alias for the string str.

Basic operation

- int `set_comm_option` (`comm_option_s` &ic)
Add a new command.
- template<class vec_t >
int `set_comm_option_vec` (size_t nic, vec_t &ic)
Add a vector containing new commands.
- int `set_parameters` (`collection` &co)
Set the parameters with a collection.
- int `set_param_help` (std::string param, std::string help)
Set one-line help text for a parameter named param.
- int `run_auto` (int argv, const char *argc[])
Automatically parse arguments to main and call interactive mode if required.

Data Fields

- std::string `tilde_string`
Desc.
- 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).

The hard-coded command objects

- **comm_option_s c_commands**
- **comm_option_s c_help**
- **comm_option_s c_quit**
- **comm_option_s c_exit**
- **comm_option_s c_license**
- **comm_option_s c_warranty**
- **comm_option_s c_set**
- **comm_option_s c_get**
- **comm_option_s c_run**
- **comm_option_s c_no_intro**
- **comm_option_s c_alias**

Static Public Attributes

Value to indicate whether commands are also command-line options

- static const int **comm_option_command** = 0
- static const int **comm_option_cl_param** = 1
- static const int **comm_option_both** = 2

Protected Member Functions

- int **expand_tilde** (std::vector< std::string > &sv)
Desc.
- int **apply_alias** (std::vector< std::string > &sv, std::string sold, std::string snw)
Replace all occurrences of sold with snw in sv.
- int **separate** (std::string str, std::vector< std::string > &sv)
Separate a string into words, handling quotes.
- bool **string_equal_dash** (std::string s1, std::string s2)
Compare two strings, treating dashes and underscores as equivalent.

The hard-coded command functions

- int **comm_option_alias** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_commands** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_get** (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_no_intro** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_run** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_set** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_warranty** (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_s > 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`

8.37.2 Member Function Documentation

8.37.2.1 virtual char* cli_gets (const char * c) [inline, virtual]

The function which obtains input from the user.

Todo

Should this be protected?

Definition at line 423 of file cli.h.

8.37.2.2 int process_args (int argv, const char * argc[], std::vector< cmd_line_arg > & ca, int debug = 0)

Process command-line arguments from a const char array.

This doesn't actually execute the functions for the corresponding options, but simply processes the parameters `argv` and `argc` and packs the information into `ca`.

8.37.2.3 int separate (std::string str, std::vector< std::string > & sv) [protected]

Separate a string into words, handling quotes.

This function separates a string into words, and handles words that begin with a " by adding more words until finding one which ends with a " .

This is used to reformat command descriptions and help text for the screen width in `comm_option_help()`, to process lines read from a file in `comm_option_run()`, and to process input in `run_interactive()`.

8.37.2.4 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 473 of file cli.h.

8.37.2.5 int set_comm_option (comm_option_s & ic)

Add a new command.

Each command/option must have either a short form in `comm_option_s::shrt` or a long form in `comm_option_s::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.

8.37.2.6 int set_verbose (int v)

Set verbosity.

Most errors are output to the screen even if verbose is zero.

8.37.3 Field Documentation**8.37.3.1 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 277 of file cli.h.

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

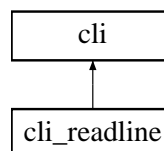
- cli.h

8.38 cli_readline Class Reference

An extension to `cli` which uses readline.

```
#include <cli_readline.h>
```

Inheritance diagram for `cli_readline::`

**8.38.1 Detailed Description**

An extension to `cli` which uses readline.

This header-only class requires the GNU `readline` library for use, but is not referenced by `O2scl` code at the moment to make the library usable without `readline`.

Definition at line 43 of file `cli_readline.h`.

Public Member Functions

- `cli_readline` (std::string fname=".cli_hist", size_t max_size=100)

Protected Attributes

- char * [line_read](#)
Buffer for readline.
- std::string [histfile](#)
String containing filename.
- size_t [msize](#)
Maximum history file size.

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

- cli_readline.h

8.39 cmd_line_arg Struct Reference

A command-line argument for [cli](#).

```
#include <cli.h>
```

8.39.1 Detailed Description

A command-line argument for [cli](#).

This is the internal structure that [cli](#) uses to package command-line arguments.

Definition at line 143 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_s](#) * [cop](#)
A pointer to the appropriate option (0, unless it's an option).

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

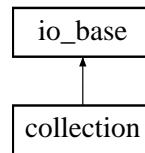
- cli.h

8.40 collection Class Reference

Collection of objects.

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

Inheritance diagram for collection::



8.40.1 Detailed Description

Collection of objects.

This class is experimental.

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 0, then both the type and the name are set to `null`.

Definition at line 488 of file `collection.h`.

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.

Public Member Functions

- int `is_parameter` (std::string tname)
Return true if there is a parameter in the *collection* with the name `tname`.

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 text file 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 `size` ()
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_void` (std::string name, `io_base` *tio, void *vec, int sz=0, int sz2=0, bool overwrt=true, bool owner=false)

- int **add_void** (std::string name, std::string stype, void *vec, int sz=0, int sz2=0, bool overwrt=true, bool owner=false)

Machine type get methods

- bool **getb_def** (std::string name, bool def_value)
Get a boolean value named name with default value def_value.
- char **getc_def** (std::string name, char def_value)
Get a char named name with default value def_value.
- double **getd_def** (std::string name, double def_value)
Get a double named name with default value def_value.
- int **geti_def** (std::string name, int def_value)
Get an integer named name with default value def_value.
- std::string **gets_def** (std::string name, std::string def_value)
Get a string named name with default value def_value.
- std::string **getw_def** (std::string name, std::string def_value)
Get a boolean value named name with default value def_value.
- bool **getb** (std::string name)
Desc.
- char **getc** (std::string name)
Desc.
- double **getd** (std::string name)
Desc.
- int **geti** (std::string name)
Desc.
- std::string **gets** (std::string name)
Desc.
- std::string **getw** (std::string name)
Desc.
- int **getb** (std::string name, bool *&op, size_t &sz)
Desc.
- int **getb** (std::string name, bool **&op, size_t &sz, size_t &sz2)
Desc.
- int **getc** (std::string name, char *&op, size_t &sz)
Desc.
- int **getc** (std::string name, char **&op, size_t &sz, size_t &sz2)
Desc.
- int **getd** (std::string name, double *&op, size_t &sz)
Desc.
- int **getd** (std::string name, double **&op, size_t &sz, size_t &sz2)
Desc.
- int **geti** (std::string name, int *&op, size_t &sz)
Desc.
- int **geti** (std::string name, int **&op, size_t &sz, size_t &sz2)
Desc.
- int **gets** (std::string name, std::string *&op, size_t &sz)
Desc.
- int **gets** (std::string name, std::string **&op, size_t &sz, size_t &sz2)
Desc.
- int **getw** (std::string name, std::string *&op, size_t &sz)
Desc.
- int **getw** (std::string name, std::string **&op, size_t &sz, size_t &sz2)
Desc.

Generic type get methods

- template<class obj_t >
int **get** (std::string name, obj_t *&op)
Desc.
- template<class obj_t >
int **get** (std::string name, obj_t *&op, size_t &sz)

Desc.

- `template<class obj_t >`
`int get (std::string name, obj_t **&op, size_t &sz, size_t &sz2)`
Desc.
- `int get_void (std::string tname, void *&vec)`
Get an object.

Text file get and set methods

- `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, bool err_on_notfound=true)`
Set object named name with input from tif.
- `int set (std::string name, std::string val, bool err_on_notfound=true)`
Set object named name with input from val.

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.

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.

8.40.2 Member Function Documentation

8.40.2.1 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.

8.40.2.2 int get_void (std::string tname, void *& vec)

Get an object.

This should be deprecated, but is presently used in `cli.cpp`

8.40.2.3 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](#).

8.40.2.4 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](#).

8.40.2.5 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](#).

8.40.2.6 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".

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

- [collection.h](#)

8.41 collection::iterator Class Reference

An [iterator](#) for stepping through a [collection](#).

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

8.41.1 Detailed Description

An [iterator](#) for stepping through a [collection](#).

Definition at line 1243 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](#)

8.42 collection::type_iterator Class Reference

An [iterator](#) for stepping through the entries in a [collection](#) of a particular type.

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

8.42.1 Detailed Description

An [iterator](#) for stepping through the entries in a [collection](#) of a particular type.

Definition at line 1299 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](#)

8.43 collection_entry Struct Reference

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

8.43.1 Detailed Description

An entry in a [collection](#)

This class is experimental.

Definition at line 73 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](#)

8.44 columnify Class Reference

Create nicely formatted columns from a [table](#) of strings.

```
#include <columnify.h>
```

8.44.1 Detailed Description

Create nicely formatted columns from a [table](#) of strings.

This is a brute-force approach of order $\text{ncols} \times \text{nrows}$.

Idea for future

Move the [screenify\(\)](#) functionality from [misc.h](#) into this class?

Definition at line 49 of file [columnify.h](#).

Public Member Functions

- `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.

8.44.2 Member Function Documentation

8.44.2.1 `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 83 of file [columnify.h](#).

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

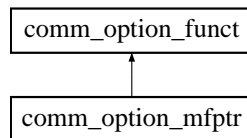
- [columnify.h](#)

8.45 comm_option_func Class Reference

Base for [cli](#) command function.

```
#include <cli.h>
```

Inheritance diagram for `comm_option_func`:



8.45.1 Detailed Description

Base for [cli](#) command function.

See the [cli](#) class for more details.

Definition at line 43 of file cli.h.

Public Member Functions

- 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:

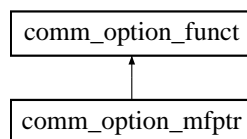
- cli.h

8.46 comm_option_mfptr Class Template Reference

Member function pointer for [cli](#) command function.

```
#include <cli.h>
```

Inheritance diagram for comm_option_mfptr::



8.46.1 Detailed Description

```
template<class tclass> class comm_option_mfptr< tclass >
```

Member function pointer for [cli](#) command function.

Definition at line 59 of file cli.h.

Public Member Functions

- [comm_option_mfptr](#) (tclass *tp, int(tclass::*fp)(std::vector< std::string > &, bool))
Create from a member function pointer from the specified class.
- virtual int [operator\(\)](#) (std::vector< std::string > &cstr, bool itive_com)
The basic function called by [cli](#).

Protected Member Functions

- `comm_option_mfptr` (const `comm_option_mfptr` &f)
Copy constructor.
- `comm_option_mfptr` & `operator=` (const `comm_option_mfptr` &f)
Copy constructor.

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`

8.47 comm_option_s Struct Reference

Command for interactive mode in `cli`.

```
#include <cli.h>
```

8.47.1 Detailed Description

Command for interactive mode in `cli`.

See the `cli` class for more details.

Definition at line 115 of file `cli.h`.

Data Fields

- `char` `shrt`
Short option (' \0' for none, must be unique if present).
- `std::string` `lng`
Long option (must be specified and must be unique).
- `std::string` `desc`
Description for help.
- `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.
- `std::string` `help`
The help description.
- `comm_option_func_t` * `func`
The pointer to the function to be called (or 0 for no function).
- `int` `type`
Type: command-line parameter, command, or both.

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

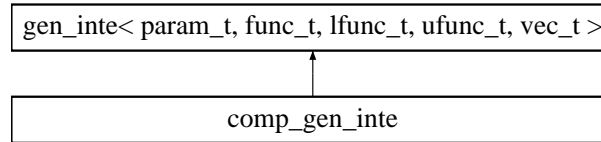
- `cli.h`

8.48 comp_gen_inte Class Template Reference

Naive generalized multi-dimensional integration.

```
#include <comp_gen_inte.h>
```

Inheritance diagram for comp_gen_inte::



8.48.1 Detailed Description

```
template<class param_t, class func_t, class lfunc_t = func_t, class ufunc_t = func_t, class vec_t = ovector_base, 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 the index specified in the function `set_oned_inte()`. For n -dimensional integration, n one-dimensional integration objects should be specified, with indexes 0 through $n-1$. The integration routines are called in order of their index, so that the outermost integration is done by the routine specified with index 0. The integral performed is:

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

This class is particularly useful if f_0 is time-consuming to evaluate, and separable from f_{n-1} .

See the discussion about the functions `func`, `lower` and `upper` in the documentation for the class [gen_inte](#).

No error estimate is performed. Error estimation for multiple dimension integrals is provided by the Monte Carlo integration classes (see [mcarlo_inte](#)).

Idea for future

Provide an example of usage for this class.

Definition at line 75 of file `comp_gen_inte.h`.

Public Member Functions

- int `set_oned_inte` (`inte`< size_t > &it, size_t i)
Set the one-dimensional integration object with index i.
- 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").

Data Fields

- size_t `max_dim`
The maximum number of integration dimensions (default 100).

Protected Member Functions

- double `odfunc` (double x, size_t &ix)
The one-dimensional integration function.

Protected Attributes

- alloc_vec_t * `cx`
The independent variable vector.
- lfunc_t * `lowerp`
The function specifying the lower limits.
- ufunc_t * `upperp`
The function specifying the upper limits.
- func_t * `mf`
The function to be integrated.
- size_t `ndim`
The number of dimensions.
- param_t * `vp`
The user-specified parameter.
- 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 >, size_t > * `fmn`
The function to send to the integrators.
- size_t `nint`
The size of the integration object arrays.
- `inte`< size_t, `funct`< size_t > > ** `iptrs`
Pointers to the integration objects.
- bool * `tptrs`
Flag indicating if integration object has been set.

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

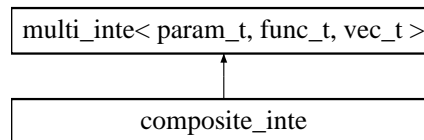
- `comp_gen_inte.h`

8.49 composite_inte Class Template Reference

Naive multi-dimensional integration over a hypercube.

```
#include <composite_inte.h>
```

Inheritance diagram for `composite_inte`:



8.49.1 Detailed Description

```
template<class param_t, class func_t = multi_func<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc> 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 1-dimensional integration routines are specified in the function [set_oned_inte\(\)](#).

The integration routines are called in order of the index specified in the function [set_oned_inte\(\)](#). For n -dimensional integration, n one-dimensional integration objects should be specified, with indexes 0 through $n-1$. The integration routines are called in order of their index, so that the outermost integration is done by the routine specified with index 0.

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

No error estimate is performed. Error estimation for multiple dimension integrals is provided by the Monte Carlo integration classes (see [mcarlo_inte](#)).

Idea for future

Create a function to set an entire array of one-dimensional integration objects at once

Definition at line 70 of file `composite_inte.h`.

Public Member Functions

- `int set_oned_inte (inte< size_t > &it, size_t i)`
Set the one-dimensional integration object with index i .
- `virtual int minteg_err (func_t &func, size_t n, const vec_t &a, const vec_t &b, param_t &pa, double &res, double &err)`
Integrate function `func` over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i < ndim-1$.
- `virtual const char * type ()`
Return string denoting type ("`composite_inte`").

Data Fields

- `size_t max_dim`
The maximum number of integration dimensions (default 100).

Protected Member Functions

- `double odfunc (double x, size_t &ix)`
The one-dimensional integration function.

Protected Attributes

- `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.
- `size_t ndim`
The user-specified number of dimensions.
- `param_t * vp`
The user-specified parameter.
- `alloc_t ao`

Memory allocator for objects of type `alloc_vec_t`.

- `func_t mfpnr_noerr` < `composite_inte` < `param_t`, `func_t`, `vec_t`, `alloc_vec_t`, `alloc_t` >, `size_t` > * `fmn`
This function to send to the integrators.
- `size_t nint`
The size of the integration object arrays.
- `inte` < `size_t` > ** `iptrs`
Pointers to the integration objects.
- `bool` * `tptrs`
Flag indicating if integration object has been set.

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

- `composite_inte.h`

8.50 contour Class Reference

Calculate `contour` lines from a two-dimensional data set.

```
#include <contour.h>
```

8.50.1 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()`

The contours are generated as a series of x- and y-coordinates, defining a line. If the `contour` is closed, then the first and the last set of coordinates will be equal.

The storage of the matrix to be specified in the function `set_data()` and this function is designed to follow the format:

$$\begin{array}{ccccc} & x_0 & x_1 & x_2 & \\ y_0 & M_{00} & M_{01} & M_{02} & \\ y_1 & M_{10} & M_{11} & M_{12} & \\ y_2 & M_{20} & M_{21} & M_{22} & \end{array}$$

thus the matrix should be $M[i][j]$ where i is the y index and j is the row index. (See also the discussion in the User's guide in the section called [Rows and columns vs. x and y](#).)

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.

Note that in order to simplify the algorithm for computing `contour` lines, the `calc_contours()` function will adjust the user-specified `contour` levels slightly in order to ensure that no `contour` line passes exactly through any data point on the grid. The contours are adjusted by multiplying the original `contour` level by 1 plus a small number (10^{-8} by default), which is specified in `lev_adjust`.

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 refines the data for any interpolation type.

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. For contours which do not close inside the region of interest, the results will always end at either

the minimum or maximum values of the x or y grid points (no extrapolation is ever done). Note also that the points defining the [contour](#) are not necessarily equally spaced, but two neighboring points will never be farther apart than the distance across opposite corners of one cell in the grid.

The Algorithm:

This works by viewing the data as defining a square two-dimensional grid. The function [calc_contours\(\)](#) exhaustively enumerates every line segment in the grid which involves a level crossing and then organizes the points defined by the intersection of a line segment with a level curve into a full [contour](#).

Idea for future

Rewrite the code which adjusts the [contour](#) levels to ensure contours don't go through the data to adjust the internal copy of the data instead.

Idea for future

It would be nice to have a function which creates a set of closed regions to fill which represent the data. However, this likely requires a completely new algorithm, because it's not easy to simply close the contours already generated by the [calc_contours\(\)](#) function. There are, for example, several cases which are difficult to handle, such as filling a region in between several closed contours.

Definition at line 164 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.
- `int calc_contours (std::vector< contour_line > &clines, bool debug=false)`
Calculate the contours.

Regrid function

- `int regrid_data (size_t xfact, size_t yfact, base_interp_mgr< ovector_const_view > &bim1, base_interp_mgr< ovector_const_subvector > &bim2)`
Regrid the data.

Obtain internal data

- `int get_data (size_t &sizex, size_t &sizey, ovector *&x_fun, ovector *&y_fun, omatrix *&udata)`
Get the data.
- `int get_edges (std::vector< edge_crossings > &rt_edges, std::vector< edge_crossings > &bm_edges)`
Return the edges.
- `int print_edges (edge_crossings &right, edge_crossings &bottom)`
Print out the edges to cout.

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, [edge_crossings](#) &right, [edge_crossings](#) &bottom)
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, [edge_crossings](#) &right, [edge_crossings](#) &bottom)
Find next point starting from a point on a bottom edge.
- int [find_intersections](#) (size_t ilev, double &level, [edge_crossings](#) &right, [edge_crossings](#) &bottom)
Find all of the intersections of the edges with the [contour](#) level.
- int [right_edges](#) (double level, o2scl::sm_interp *si, [edge_crossings](#) &right)
Interpolate all right edge crossings.
- int [bottom_edges](#) (double level, o2scl::sm_interp *si, [edge_crossings](#) &bottom)
Interpolate all bottom edge crossings.
- int [process_line](#) (int j, int k, int dir, [ovector](#) &x, [ovector](#) &y, bool first, [edge_crossings](#) &right, [edge_crossings](#) &bottom)
Create a [contour](#) line from a starting edge.
- int [check_data](#) ()
Check to ensure the x- and y-arrays are monotonic.

Protected Attributes

- std::vector< [edge_crossings](#) > [red](#)
Right edge list.
- std::vector< [edge_crossings](#) > [bed](#)
Bottom edge list.

User-specified data

- int [nx](#)
- int [ny](#)
- [ovector](#) [xfun](#)
- [ovector](#) [yfun](#)
- [omatrix](#) [data](#)

User-specified contour levels

- int [nlev](#)
- [ovector](#) [levels](#)
- bool [levels_set](#)

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

8.50.2 Member Function Documentation

8.50.2.1 `int calc_contours (std::vector< contour_line > & clines, bool debug = false)`

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.

8.50.2.2 `int get_data (size_t & sizex, size_t & sizey, ovector *& x_fun, ovector *& y_fun, omatrix *& udata)` [inline]

Get the data.

This is useful to see how the data has changed after a call to `regrid_data()`.

Idea for future

There is probably a better way than returning pointers to the internal data.

Definition at line 277 of file `contour.h`.

8.50.2.3 `int regrid_data (size_t xfact, size_t yfact, base_interp_mgr< ovector_const_view > & bim1, base_interp_mgr< ovector_const_subvector > & bim2)`

Regrid the data.

Use interpolation to refine the data set. This can be called before `calc_contours()` in order to make the `contour` levels smoother by providing a smaller grid size. If the original number of data points is (n_x, n_y) , then the new number of data points is

$$(xfact (n_x - 1) + 1, yfact (n_y - 1) + 1)$$

The parameters `xfact` and `yfact` must both be larger than zero and they cannot both be 1.

8.50.2.4 `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 187 of file `contour.h`.

8.50.2.5 `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 218 of file `contour.h`.

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

- `contour.h`

8.51 `contour_line` Class Reference

A `contour` line.

```
#include <contour.h>
```

8.51.1 Detailed Description

A `contour` line.

The `contour` lines generated by the `contour` class are given as objects of this type.

Idea for future

Write an I/O object for `contour` lines

Idea for future

Make this a subclass of `contour` .

Definition at line 43 of file `contour.h`.

Public Member Functions

- `contour_line` ()
Create an empty line.
- `contour_line` (const `contour_line` &c)
Copy constructor for STL allocator.

Data Fields

- double `level`
The `contour` level.
- `ovector` x
The line x coordinates.
- `ovector` y
The line y coordinates.

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

- `contour.h`

8.52 `convert_units` Class Reference

Convert units.

```
#include <convert_units.h>
```

8.52.1 Detailed Description

Convert units.

Allow the user to convert between two different units after specifying a conversion factor. This class will also automatically combine two conversion factors to create a new unit conversion.

Conversions are performed by the [convert\(\)](#) function and the conversion factors must be specified beforehand using the [insert_cache\(\)](#) function.

Example:

```
convert_units cu;
cu.insert_cache("in", "cm", 2.54);
cout << "12 in is " << cu.convert("in", "cm", 12.0) << " cm. " << endl;
```

Idea for future

Ideally, a real C++ API for the GNU units command would probably be better.

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

Data Structures

- struct [unit_t](#)
The type for caching unit conversions.

Public Member Functions

- virtual double [convert](#) (std::string from, std::string to, double val)
Return the value val after converting using units from and to.
- int [insert_cache](#) (std::string from, std::string to, double conv)
Manually insert a unit conversion into the cache.
- int [remove_cache](#) (std::string from, std::string to)
Manually remove a unit conversion into the cache.
- int [print_cache](#) ()
Print the present unit cache to std::cout.
- int [energy_conv](#) ()
Add conversion factors for energy equivalents.

Data Fields

- int [verbose](#)
Verbosity (default 0).
- bool [use_gnu_units](#)
(default false)
- bool [err_on_fail](#)
(default true)
- std::string [units_cmd_string](#)
Default 'units'.

Protected Types

- typedef std::map< std::string, [unit_t](#), [string_comp](#) >::iterator [miter](#)
The iterator type.

Protected Attributes

- std::map< std::string, [unit_t](#), [string_comp](#) > [mcache](#)
The cache where unit conversions are stored.

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

- `convert_units.h`

8.53 convert_units::unit_t Struct Reference

The type for caching unit conversions.

```
#include <convert_units.h>
```

8.53.1 Detailed Description

The type for caching unit conversions.

Definition at line 70 of file convert_units.h.

Data Fields

- std::string [f](#)
The input unit.
- std::string [t](#)
The output unit.
- double [c](#)
The conversion factor.

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

- convert_units.h

8.54 coutput Class Reference

Class to control object output.

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

8.54.1 Detailed Description

Class to control object output.

This class is experimental.

Definition at line 1475 of file collection.h.

Data Structures

- struct [ltptr](#)
Order the pointers by numeric value.

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`.

8.54.2 Member Function Documentation

8.54.2.1 `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`

8.54.3 Field Documentation

8.54.3.1 `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 1531 of file `collection.h`.

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

- `collection.h`

8.55 coutput::ltptr Struct Reference

Order the pointers by numeric value.

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

8.55.1 Detailed Description

Order the pointers by numeric value.

Definition at line 1497 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:

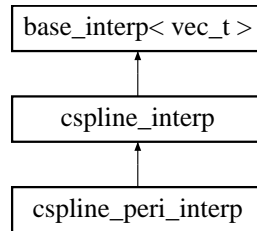
- [collection.h](#)

8.56 cspline_interp Class Template Reference

Cubic spline interpolation (GSL).

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

Inheritance diagram for cspline_interp::



8.56.1 Detailed Description

template<class vec_t> class cspline_interp< vec_t >

Cubic spline interpolation (GSL).

By default, this uses natural boundary conditions, where the second derivative vanishes at each end point. Extrapolation effectively assumes that the second derivative is linear outside of the endpoints. If the boolean argument to the constructor is set to `true`, then periodic boundary conditions are assumed. Alternatively, use [cspline_peri_interp](#) for periodic boundary conditions.

Idea for future

Could use `O2scl`'s native tridiagonal routines instead of calling the GSL ones.

Definition at line 307 of file `interp.h`.

Public Member Functions

- [cspline_interp](#) (bool periodic=false)
Create a base interpolation object with natural or periodic boundary conditions.
- virtual int [allocate](#) (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 $\int_a^b y(x) dx$.
- virtual const char * [type](#) ()
Return the type, "cspline_interp".

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.

Protected Attributes

- bool [peri](#)
True for periodic boundary conditions.

Storage for cubic spline interpolation

- double * **c**
- double * **g**
- double * **diag**
- double * **offdiag**

Private Member Functions

- [cspline_interp](#) (const [cspline_interp](#)< vec_t > &)
- [cspline_interp](#)< vec_t > & **operator=** (const [cspline_interp](#)< vec_t > &)

8.56.2 Member Function Documentation

8.56.2.1 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 390 of file interp.h.

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

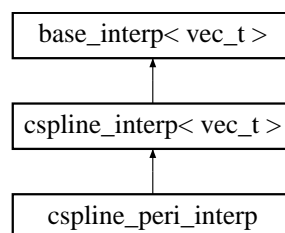
- interp.h

8.57 cspline_peri_interp Class Template Reference

Cubic spline interpolation with periodic boundary conditions (GSL).

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

Inheritance diagram for cspline_peri_interp::



8.57.1 Detailed Description

template<class vec_t> class cspline_peri_interp< vec_t >

Cubic spline interpolation with periodic boundary conditions (GSL).

This is convenient to allow interpolation objects to be supplied as template parameters

Definition at line 692 of file interp.h.

Public Member Functions

- virtual const char * [type](#) ()
Return the type, "cspline_peri_interp".

Private Member Functions

- [cspline_peri_interp](#) (const [cspline_peri_interp](#)< vec_t > &)
- [cspline_peri_interp](#)< vec_t > & [operator=](#) (const [cspline_peri_interp](#)< vec_t > &)

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

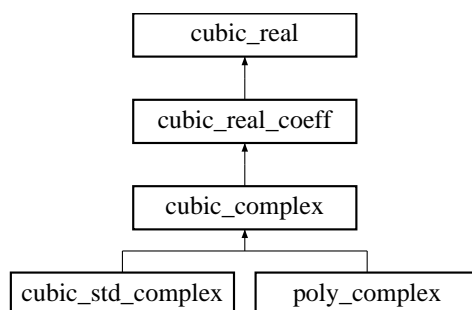
- interp.h

8.58 cubic_complex Class Reference

Solve a cubic polynomial with complex coefficients and complex roots [abstract base].

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

Inheritance diagram for cubic_complex::



8.58.1 Detailed Description

Solve a cubic polynomial with complex coefficients and complex roots [abstract base].

Definition at line 189 of file poly.h.

Public Member Functions

- 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)=0
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 $x = x_3$.
- const char * [type](#) ()
Return a string denoting the type ("cubic_complex").

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

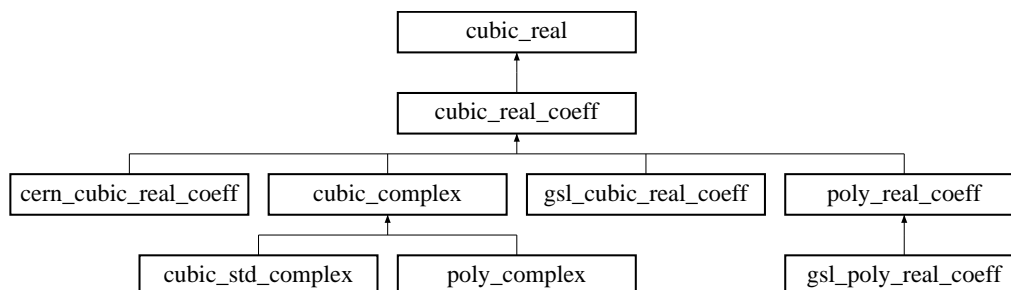
- [poly.h](#)

8.59 cubic_real Class Reference

Solve a cubic polynomial with real coefficients and real roots [abstract base].

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

Inheritance diagram for cubic_real::



8.59.1 Detailed Description

Solve a cubic polynomial with real coefficients and real roots [abstract base].

Definition at line 138 of file poly.h.

Public Member Functions

- virtual int [solve_r](#) (const double a3, const double b3, const double c3, const double d3, double &x1, double &x2, double &x3)=0
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:

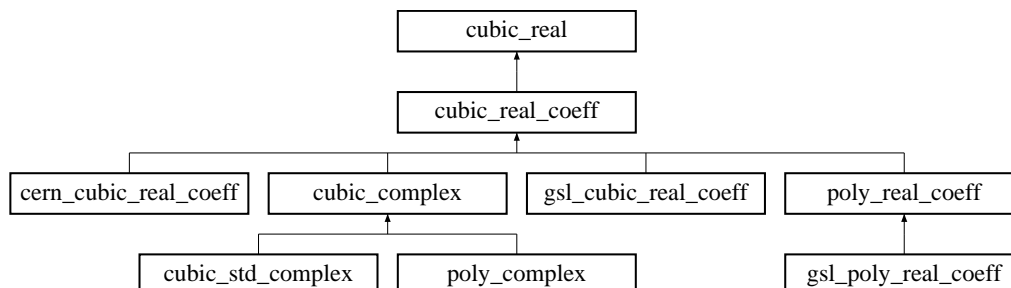
- [poly.h](#)

8.60 cubic_real_coeff Class Reference

Solve a cubic polynomial with real coefficients and complex roots [abstract base].

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

Inheritance diagram for cubic_real_coeff::



8.60.1 Detailed Description

Solve a cubic polynomial with real coefficients and complex roots [abstract base].

Definition at line 159 of file poly.h.

Public Member Functions

- 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)=0
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:

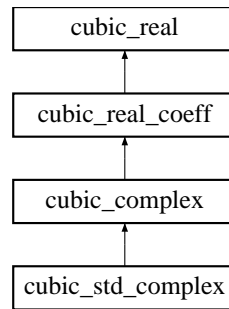
- [poly.h](#)

8.61 cubic_std_complex Class Reference

Solve a cubic with complex coefficients and complex roots.

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

Inheritance diagram for cubic_std_complex::



8.61.1 Detailed Description

Solve a cubic with complex coefficients and complex roots.

Definition at line 669 of file poly.h.

Public Member Functions

- 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)
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 $x = x_3$.
- const char * [type](#) ()
Return a string denoting the type ("cubic_std_complex").

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

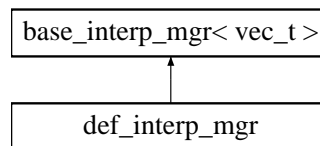
- [poly.h](#)

8.62 def_interp_mgr Class Template Reference

A base interpolation object manager template.

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

Inheritance diagram for def_interp_mgr::



8.62.1 Detailed Description

```
template<class vec_t, template< class > class interp_t> class def_interp_mgr< vec_t, interp_t >
```

A base interpolation object manager template.

Definition at line 1053 of file interp.h.

Public Member Functions

- virtual [base_interp](#)< vec_t > * [new_interp](#) ()
Create a new interpolation object.

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

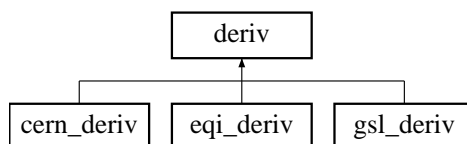
- [interp.h](#)

8.63 deriv Class Template Reference

Numerical differentiation base [abstract base].

```
#include <deriv.h>
```

Inheritance diagram for deriv::



8.63.1 Detailed Description

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

Numerical differentiation base [abstract 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](#).

Data Structures

- struct [dpars](#)
A structure for passing the function to second and third derivatives.

Public Member Functions

- 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)=0
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

- bool `err_nonconv`
If true, call the error handler if the routine does not "converge".
- 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)=0
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.

8.63.2 Member Function Documentation

8.63.2.1 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 93 of file deriv.h.

8.63.2.2 virtual int calc_err_int (double x, dpars &pa, o2scl::funct< dpars > &func, double &dfdx, double &err) [protected, pure 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

8.63.2.3 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 174 of file `deriv.h`.

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

- `deriv.h`

8.64 deriv::dpars Struct Reference

A structure for passing the function to second and third derivatives.

```
#include <deriv.h>
```

8.64.1 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:

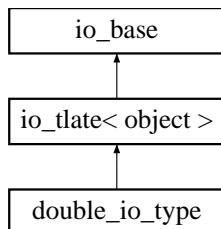
- `deriv.h`

8.65 double_io_type Class Reference

I/O object for double variables.

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

Inheritance diagram for `double_io_type`:



8.65.1 Detailed Description

I/O object for double variables.

This class is experimental.

Definition at line 2289 of file `collection.h`.

Public Member Functions

- [`double_io_type`](#) (const char *)
Desc.

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

- [`collection.h`](#)

8.66 `edge_crossings` Class Reference

Edges for the [`contour`](#) class.

```
#include <contour.h>
```

8.66.1 Detailed Description

Edges for the [`contour`](#) class.

The edge crossings generated by the [`contour`](#) class are given as objects of this type.

Idea for future

Write an I/O object for edge crossings

Idea for future

Make this a subclass of [`contour`](#) .

Definition at line 73 of file `contour.h`.

Data Fields

- [`omatrix_int`](#) status
Edge status.
- [`omatrix`](#) values
Edge values.

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

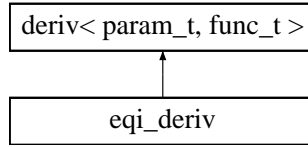
- `contour.h`
-

8.67 eqi_deriv Class Template Reference

Derivatives for equally-spaced abscissas.

```
#include <eqi_deriv.h>
```

Inheritance diagram for eqi_deriv::



8.67.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_base> 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 `xxx` (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:

$$f'(x_0 + ph) = \frac{1}{4h^2} [(1 - 2p)^2 f_{-2} + (8p - 16p^2) f_{-1} - (2 - 24p^2) f_0 - (8p + 16p^2) f_1 + (1 + 2p)^2 f_2]$$

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

- 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 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.
- virtual int [calc2_err](#) (double x, param_t &pa, func_t &func, double &dfdx, double &err)
Calculate the second derivative of func w.r.t. x.
- virtual int [calc3_err](#) (double x, param_t &pa, func_t &func, double &dfdx, double &err)
Calculate the third derivative of func w.r.t. x.
- double [calc_vector](#) (double x, double x0, double dx, size_t nx, const vec_t &y)
Calculate the derivative at x given an array.
- double [calc2_vector](#) (double x, double x0, double dx, size_t nx, const vec_t &y)
Calculate the second derivative at x given an array.
- double [calc3_vector](#) (double x, double x0, double dx, size_t nx, const vec_t &y)
Calculate the third derivative at x given an array.
- int [deriv_vector](#) (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).

8.67.2 Member Function Documentation

8.67.2.1 double calc2_vector (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 function specified in an array *y* of size *nx* with equally spaced abscissas. The first abscissa should be given as *x0* 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 259 of file eqi_deriv.h.

8.67.2.2 double calc3_vector (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 *x0* 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 277 of file eqi_deriv.h.

8.67.2.3 double calc_vector (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 function specified in an array *y* of size *nx* with equally spaced abscissas. The first abscissa should be given as *x0* 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.

Todo

Document or change the value of 100.0 which appears here

Definition at line 242 of file eqi_deriv.h.

8.67.2.4 int deriv_vector (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 *np*oints.

Definition at line 291 of file eqi_deriv.h.

8.67.2.5 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.

8.67.2.6 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`.

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

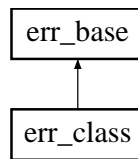
- `eqi_deriv.h`

8.68 `err_base` Class Reference

Class defining an error handler [abstract base].

```
#include <err_hnd.h>
```

Inheritance diagram for `err_base`:



8.68.1 Detailed Description

Class defining an error handler [abstract base].

A global object of this type is defined, [err_hnd](#).

Idea for future

There may be an issue associated with the string manipulations causing errors in the error handler.

Definition at line 142 of file `err_hnd.h`.

Public Member Functions

- virtual void [set](#) (const char *reason, const char *file, int line, int lerrno)=0
Set an error.
- virtual void [get](#) (const char *&reason, const char *&file, int &line, int &lerrno)=0
Get the last error.
- virtual int [get_errno](#) ()=0
Return the last error number.
- virtual int [get_line](#) ()=0
Return the line number of the last error.
- virtual const char * [get_reason](#) ()=0
Return the reason for the last error.
- virtual const char * [get_file](#) ()=0
Return the file name of the last error.
- virtual const char * [get_str](#) ()=0
Return a string summarizing the last error.
- virtual void [reset](#) ()=0
Remove last error information.

Static Public Member Functions

- static void [gsl_hnd](#) (const char *reason, const char *file, int line, int lerrno)
Set an error.

8.68.2 Member Function Documentation

8.68.2.1 `static void gsl_hnd (const char *reason, const char *file, int line, int lerrno)` `[inline, static]`

Set an error.

This is separate from `set()`, since the gsl error handler needs to be a static function.

Definition at line 158 of file `err_hnd.h`.

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

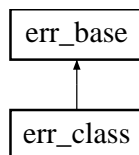
- [err_hnd.h](#)

8.69 `err_class` Class Reference

The error handler.

```
#include <err_hnd.h>
```

Inheritance diagram for `err_class`:



8.69.1 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 200 of file `err_hnd.h`.

Public Member Functions

- virtual void `set` (const char *reason, const char *file, int line, int lerrno)
Set an error.
- virtual void `get` (const char *&reason, const char *&file, int &line, int &lerrno)
Get the last error.
- virtual int `get_errno` ()
Return the last error number.
- virtual int `get_line` ()
Return the line number of the last error.
- virtual const char * `get_reason` ()
Return the reason for the last error.
- virtual const char * `get_file` ()
Return the file name of the last error.
- virtual const char * `get_str` ()
Return a string summarizing the last error.
- virtual void `reset` ()
Remove last error information.
- void `set_mode` (int m)

Set error handling mode.

- `int get_mode ()`
Return the error handling mode.

Data Fields

- `bool array_abort`
If true, call `exit()` when an array index error is set (default true).
- `size_t fname_size`
Number of characters from filename to print (default 35).

Protected Attributes

- `int a_errno`
The error number.
- `int a_line`
The line number.
- `int mode`
The mode of error handling (default 2).
- `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 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.

8.69.2 Member Function Documentation

8.69.2.1 `void set_mode (int m)` [inline]

Set error handling mode.

- 0 - Continue execution after an error occurs
- 1 - Continue execution after an error occurs
- 2 - Abort execution after an error occurs, and print out the error information (default)

Definition at line 241 of file `err_hnd.h`.

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

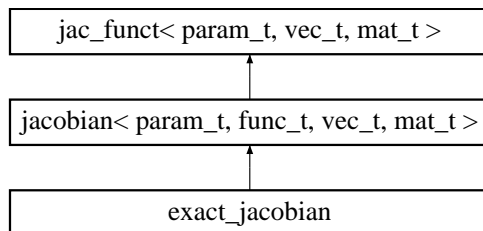
- `err_hnd.h`
-

8.70 exact_jacobian Class Template Reference

A direct calculation of the [jacobian](#) using a [deriv](#) object.

```
#include <jacobian.h>
```

Inheritance diagram for `exact_jacobian`::



8.70.1 Detailed Description

```
template<class param_t, class func_t = mm_funct<param_t>, class vec_t = ovector_base, class mat_t = omatrix_base> 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. This class is mostly useful for demonstration purposes.

Definition at line 545 of file `jacobian.h`.

Data Structures

- struct [ej_parms](#)
Parameter structure for passing information.

Public Member Functions

- 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)
Function for the derivative object.

Protected Attributes

- [deriv](#)< [ej_parms](#), [funct](#)< [ej_parms](#) > > * [dptr](#)

Pointer to the derivative object.

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

- `jacobian.h`

8.71 `exact_jacobian::ej_parms` Struct Reference

Parameter structure for passing information.

```
#include <jacobian.h>
```

8.71.1 Detailed Description

```
template<class param_t, class func_t = mm_func<param_t>, class vec_t = ovector_base, class mat_t = omatrix_base>
struct exact_jacobian< param_t, func_t, vec_t, mat_t >::ej_parms
```

Parameter structure for passing information.

This class is primarily useful for specifying derivatives for using the `jacobian::set_deriv()` function.

Definition at line 567 of file `jacobian.h`.

Data Fields

- `size_t nv`
The number of variables.
- `size_t xj`
The current x value.
- `size_t yi`
The current y value.
- `vec_t * x`
The x vector.
- `vec_t * y`
The y vector.
- `param_t * pa`
The parameters.

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

- `jacobian.h`

8.72 `file_detect` Class Reference

Read a (possibly compressed) file and automatically detect the file format.

```
#include <file_detect.h>
```

8.72.1 Detailed Description

Read a (possibly compressed) file and automatically detect the file format.

This class is experimental.

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.

Idea for future

Use the boost pipes facility instead.

Definition at line 61 of file `file_detect.h`.

Public Member Functions

- `in_file_format * open` (const char *s, bool err_on_fail=true)
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.

8.72.2 Member Function Documentation

8.72.2.1 in_file_format* open (const char *s, bool err_on_fail = true)

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.

If the file opening failed, zero will be returned.

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

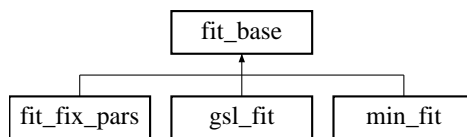
- `file_detect.h`

8.73 fit_base Class Template Reference

Non-linear least-squares fitting [abstract base].

```
#include <fit_base.h>
```

Inheritance diagram for fit_base::



8.73.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_base, class mat_t = omatrix_base> class fit_base< param_t,
func_t, vec_t, mat_t >
```

Non-linear least-squares fitting [abstract base].

Definition at line 357 of file fit_base.h.

Public Member Functions

- 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)=0
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.

8.73.2 Member Function Documentation

8.73.2.1 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) [pure 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 χ^2 is returned in chi2.

Implemented in [gsl_fit](#), [min_fit](#), and [gsl_fit< param_t, fit_func_t_mfptr< fit_fix_pars< param_t, bool_vec_t >, param_t > >](#).

8.73.2.2 `virtual int print_iter (size_t nv, 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 376 of file `fit_base.h`.

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

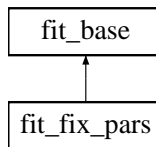
- `fit_base.h`

8.74 fit_fix_pars Class Template Reference

Multidimensional fitting fixing some parameters and varying others.

```
#include <fit_fix.h>
```

Inheritance diagram for `fit_fix_pars`:



8.74.1 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 38 of file `fit_fix.h`.

Public Member Functions

- `fit_fix_pars ()`
Specify the member function pointer.
- `virtual int fit (size_t ndat, ovector_base &xdat, ovector_base &ydat, ovector_base &yerr, size_t npar, ovector_base &par, omatrix_base &covar, double &chi2, param_t &pa, fit_funct< 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_base &xdat, ovector_base &ydat, ovector_base &yerr, size_t npar, ovector_base &par, bool_vec_t &fix, omatrix_base &covar, double &chi2, param_t &pa, fit_funct< param_t > &fitfun)`
Fit function func while fixing some parameters as specified in fix.
- `int set_fit (fit_base< param_t, fit_funct_mfptr< fit_fix_pars, param_t > > &fitter)`
Change the base minimizer.

Data Fields

- `gsl_fit< param_t, fit_funct_mfptr< fit_fix_pars, param_t > > def_fit`
The default base minimizer.

Protected Member Functions

- virtual int `fit_func` (size_t nv, `ovector_base` &x, double xx, double &y, param_t &pa)
The new function to send to the minimizer.

Protected Attributes

- `fit_base`< param_t, `fit_func_ptr`< `fit_fix_pars`, param_t > > * `fitp`
The minimizer.
- `fit_func`< 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_base` * `xp`
The user-specified initial vector.

Private Member Functions

- `fit_fix_pars` (const `fit_fix_pars` &)
- `fit_fix_pars` & `operator=` (const `fit_fix_pars` &)

8.74.2 Member Function Documentation

8.74.2.1 virtual int fit (size_t ndat, `ovector_base` &xdat, `ovector_base` &ydat, `ovector_base` &yerr, size_t npar, `ovector_base` &par, `omatrix_base` &covar, double &chi2, param_t &pa, `fit_func`< 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 χ^2 is returned in `chi2`.

Definition at line 62 of file `fit_fix.h`.

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

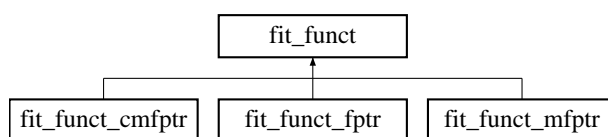
- `fit_fix.h`

8.75 fit_func Class Template Reference

Fitting function [abstract base].

```
#include <fit_base.h>
```

Inheritance diagram for `fit_func`::



8.75.1 Detailed Description

template<class param_t, class vec_t = ovector_base> class fit_funct< param_t, vec_t >

Fitting function [abstract base].

Definition at line 38 of file fit_base.h.

Public Member Functions

- virtual int **operator()** (size_t np, vec_t &p, double x, double &y, param_t &pa)=0
Using parameters in p, predict y given x.

Private Member Functions

- **fit_funct** (const **fit_funct** &)
- **fit_funct** & **operator=** (const **fit_funct** &)

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

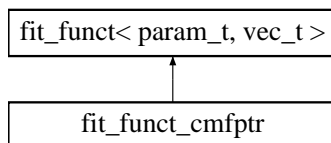
- fit_base.h

8.76 fit_funct_cmfprr Class Template Reference

Const member function pointer fitting function.

#include <fit_base.h>

Inheritance diagram for fit_funct_cmfprr::



8.76.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base> class fit_funct_cmfprr< tclass, param_t, vec_t >

Const member function pointer fitting function.

Definition at line 148 of file fit_base.h.

Public Member Functions

- **fit_funct_cmfprr** (tclass *tp, int(tclass::*fp)(size_t np, vec_t &p, double x, double &y, param_t &pa) const)
Specify the member function pointer.
- 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) const`
Storage for the user-specified function pointer.
- `tclass * tptr`
Storage for the class pointer.

Private Member Functions

- `fit_func_ptr(const fit_func_ptr &)`
- `fit_func_ptr & operator=(const fit_func_ptr &)`

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

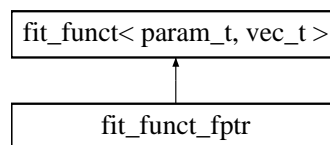
- `fit_base.h`

8.77 fit_func_ptr Class Template Reference

Function pointer fitting function.

```
#include <fit_base.h>
```

Inheritance diagram for `fit_func_ptr`:



8.77.1 Detailed Description

```
template<class param_t, class vec_t = ovector_base> class fit_func_ptr< param_t, vec_t >
```

Function pointer fitting function.

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

Public Member Functions

- `fit_func_ptr(int(*fp)(size_t np, vec_t &p, double x, double &y, param_t &pa))`
Specify a fitting function by a function pointer.
- `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_func_ptr(const fit_func_ptr &)`
- `fit_func_ptr & operator=(const fit_func_ptr &)`

Protected Attributes

- `int(* fptr)(size_t np, vec_t &p, double x, double &y, param_t &pa)`
Storage for the user-specified function pointer.

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

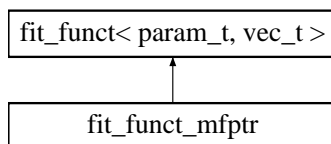
- `fit_base.h`

8.78 fit_funcnt_mfptr Class Template Reference

Member function pointer fitting function.

```
#include <fit_base.h>
```

Inheritance diagram for `fit_funcnt_mfptr`:



8.78.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_base> class fit_funcnt_mfptr< tclass, param_t, vec_t >
```

Member function pointer fitting function.

Definition at line 104 of file `fit_base.h`.

Public Member Functions

- `fit_funcnt_mfptr(tclass *tp, int(tclass::*fp)(size_t np, vec_t &p, double x, double &y, param_t &pa))`
Specify the member function pointer.
- `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_funcnt_mfptr(const fit_funcnt_mfptr &)`
- `fit_funcnt_mfptr & operator=(const fit_funcnt_mfptr &)`

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

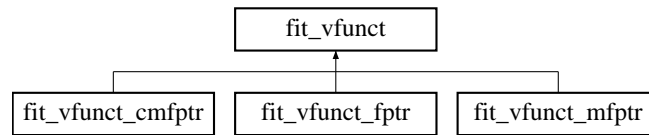
- `fit_base.h`

8.79 fit_vfunct Class Template Reference

Fitting function with arrays [abstract base].

```
#include <fit_base.h>
```

Inheritance diagram for fit_vfunct::



8.79.1 Detailed Description

```
template<class param_t, size_t nvar> class fit_vfunct< param_t, nvar >
```

Fitting function with arrays [abstract base].

Definition at line 196 of file fit_base.h.

Public Member Functions

- virtual int [operator\(\)](#) (size_t np, double p[], double x, double &y, param_t &pa)=0
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:

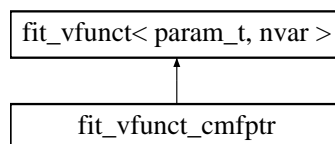
- fit_base.h

8.80 fit_vfunct_cmfptra Class Template Reference

Const member function pointer fitting function with arrays.

```
#include <fit_base.h>
```

Inheritance diagram for fit_vfunct_cmfptra::



8.80.1 Detailed Description

template<class tclass, class param_t, size_t nvar> class fit_vfunct_cmfptr< tclass, param_t, nvar >

Const member function pointer fitting function with arrays.

Definition at line 308 of file fit_base.h.

Public Member Functions

- [fit_vfunct_cmfp](#)tr (tclass *tp, int(tclass::*fp)(size_t np, double p[], double x, double &y, param_t &pa) const)
Specify the member function pointer.
- 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) const
Storage for the user-specified function pointer.
- tclass * [tptr](#)
Storage for the class pointer.

Private Member Functions

- [fit_vfunct_cmfp](#)tr (const [fit_vfunct_cmfp](#)tr &)
- [fit_vfunct_cmfp](#)tr & [operator=](#) (const [fit_vfunct_cmfp](#)tr &)

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

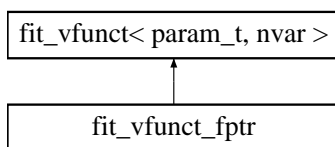
- fit_base.h

8.81 fit_vfunct_fptr Class Template Reference

Function pointer fitting function with arrays.

```
#include <fit_base.h>
```

Inheritance diagram for fit_vfunct_fptr::



8.81.1 Detailed Description

template<class param_t, size_t nvar> class fit_vfunct_fptr< param_t, nvar >

Function pointer fitting function with arrays.

Definition at line 220 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 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(* [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:

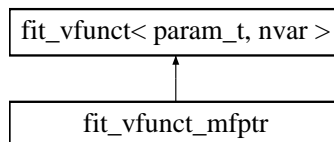
- fit_base.h

8.82 fit_vfunct_mfptr Class Template Reference

Member function pointer fitting function with arrays.

```
#include <fit_base.h>
```

Inheritance diagram for fit_vfunct_mfptr::



8.82.1 Detailed Description

```
template<class tclass, class param_t, size_t nvar> class fit_vfunct_mfptr< tclass, param_t, nvar >
```

Member function pointer fitting function with arrays.

Definition at line 263 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 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`

8.83 `format_float` Class Reference

Format a floating point number into a Latex or HTML string.

```
#include <format_float.h>
```

8.83.1 Detailed Description

Format a floating point number into a Latex or HTML string.

This class formats floating point strings into something useful for HTML or Latex documents.

The base-10 logarithm of the smallest and largest numbers to be represented without a string of the form

$$\$ \times 10^{\{\mathrm{x}\}} \$$$

can be specified in `set_exp_min()` and `set_exp_max()`. The number of significant figures can be specified with `set_sig_figs\(\)` (the default is 5).

Note that this function does not warn the user if the number of significant figures requested is larger than the machine precision.

The format for a normal number is

```
prefix (sign-string) number suffix
```

and in scientific notation is

```
sci-prefix (sci-sign-string) number times-string
exp-prefix (exp-sign-string) exponent exp-suffix sci-suffix
```

Idea for future

Implement padding with zeros

Idea for future

Separate out decimal point and make it separate.

Definition at line 78 of file `format_float.h`.

Public Member Functions

- `int html_mode ()`
Set HTML mode.
- `int latex_mode ()`
Set Latex mode.
- `std::string convert (double x)`
Convert a floating point number to a string.

Set text settings

These are modified by the functions `html_mode()` and `latex_mode()`

- `int set_prefix (std::string prefix)`
set prefix
- `int set_suffix (std::string suffix)`
Set suffix.
- `int set_sci_prefix (std::string sci_prefix)`
Set prefix for scientific notation.
- `int set_sci_suffix (std::string sci_suffix)`
Set suffix for scientific notation.
- `int set_exp_prefix (std::string exp_prefix)`
Set prefix for exponent.
- `int set_exp_suffix (std::string exp_suffix)`
Set suffix for exponent.
- `int set_sign (std::string sign)`
Set sign.
- `int set_exp_sign (std::string exp_sign)`
Set sign for exponent.
- `int set_sci_sign (std::string sci_sign)`
Set sign for scientific notation.
- `int set_times (std::string times)`
Set times.
- `int set_zero (std::string zero)`
Set zero.
- `int set_not_finite (std::string not_finite)`
Set string for numbers which are not finite.

Set other settings

These are not modified by the functions `html_mode()` and `latex_mode()`

- `int set_exp_limits (int min, int max)`
Set the exponent limits.
- `int set_sig_figs (size_t sig_figs)`
Set the number of significant figures.

Get text settings

These are modified by the functions `html_mode()` and `latex_mode()`

- `std::string get_prefix ()`
Get prefix.
- `std::string get_suffix ()`
Get suffix.
- `std::string get_sci_prefix ()`
Get prefix for scientific notation.
- `std::string get_sci_suffix ()`
Get suffix for scientific notation.
- `std::string get_exp_prefix ()`
Get prefix for exponent.
- `std::string get_exp_suffix ()`

- `std::string get_sign ()`
Get suffix for exponent.
- `std::string get_exp_sign ()`
Get sign.
- `std::string get_exp_sign ()`
Get sign for exponent.
- `std::string get_sci_sign ()`
Get sign for scientific notation.
- `std::string get_times ()`
Get times.
- `std::string get_zero ()`
Get zero.
- `std::string get_not_finite ()`
Get string for numbers which are not finite.

Get other settings

These are not modified by the functions `html_mode()` and `latex_mode()`

- `int get_exp_min ()`
Get minimum exponent.
- `int get_exp_max ()`
Get maximum exponent.
- `size_t get_sig_figs ()`
Get sig_figs.

Protected Attributes

Base text settings

- `std::string prefix`
Prefix (default "").
- `std::string suffix`
Suffix (default "").
- `std::string sgn`
Sign string (default "-").
- `std::string sci_sgn`
Sign string in scientific mode (default "-").
- `std::string exp_sgn`
Sign string for exponent in scientific mode (default "-").
- `std::string sci_prefix`
Prefix in scientific mode (default "").
- `std::string sci_suffix`
Suffix in scientific mode (default "").
- `std::string exp_prefix`
Exponent prefix (default "").
- `std::string exp_suffix`
Exponent suffix (default "").
- `std::string times`
Times symbol for scientific mode (default "x").
- `std::string not_finite`
String for numbers which are not finite (default "Nan").
- `std::string zeros`
String for zeros (default "0").

Other settings

- `size_t sig_figs`
Number of significant figures (default 5).
- `int ex_mn`
Lower limit for automatic mode (default -2).
- `int ex_mx`
Upper limit for automatic mode (default -2).
- `bool pad_zeros`
If true, pad with zeros (default false).

8.83.2 Member Function Documentation

8.83.2.1 int html_mode () [inline]

Set HTML mode.

This function is equivalent to the settings:

```
format_float::set_prefix("");
format_float::set_sign("-");
format_float::set_suffix("");
format_float::set_sci_prefix("");
format_float::set_times(" &times; ");
format_float::set_exp_prefix("10<sup>");
format_float::set_exp_sign("-");
format_float::set_sci_sign("-");
format_float::set_exp_suffix("</sup>");
format_float::set_sci_suffix("");
format_float::set_not_finite("Nan");
format_float::set_zero("0");
```

Definition at line 161 of file format_float.h.

8.83.2.2 int latex_mode () [inline]

Set Latex mode.

This function is equivalent to the settings:

```
format_float::set_prefix("");
format_float::set_sign("$-$");
format_float::set_suffix("");
format_float::set_sci_prefix("");
format_float::set_times(" $\backslash$times ");
format_float::set_exp_prefix("10^{");
format_float::set_exp_sign("-");
format_float::set_sci_sign("$-$");
format_float::set_exp_suffix("}");
format_float::set_sci_suffix("");
format_float::set_not_finite("Nan");
format_float::set_zero("0");
```

Definition at line 195 of file format_float.h.

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

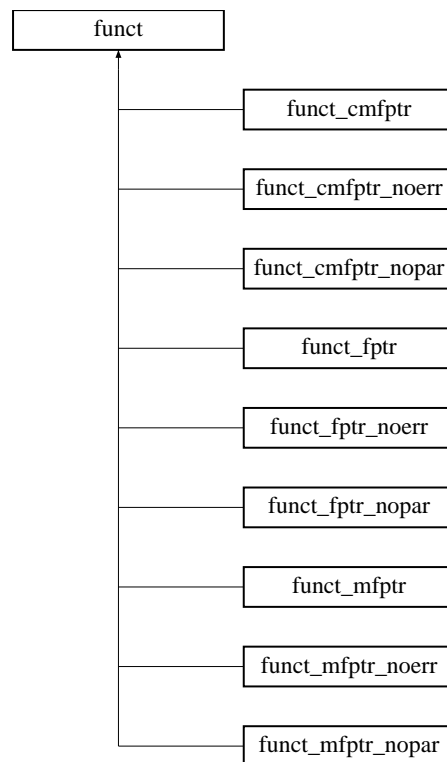
- format_float.h

8.84 funct Class Template Reference

One-dimensional function [abstract base].

```
#include <funct.h>
```

Inheritance diagram for funct::



8.84.1 Detailed Description

template<class param_t> class funct< param_t >

One-dimensional function [abstract base].

This class generalizes a function $y(x)$.

This class is one of a large number of function object classes in O₂scl designed to provide a mechanism for the user to supply functions to solvers, minimizers, integrators, etc. See [Function Objects](#) for a general description.

Definition at line 44 of file funct.h.

Public Member Functions

- virtual int [operator\(\)](#) (double x, double &y, param_t &pa)=0
Compute the function at point x, with result y.
- virtual double [operator\(\)](#) (double x, param_t &pa)
Compute the function at point x, returning the result.

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

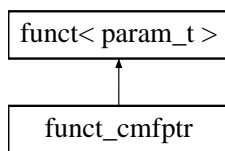
- funct.h

8.85 **funct_cmfpnr Class Template Reference**

Const member function pointer to a one-dimensional function.

```
#include <funct.h>
```

Inheritance diagram for funct_cmfptr::



8.85.1 Detailed Description

template<class tclass, class param_t> class funct_cmfptr< tclass, param_t >

Const member function pointer to a one-dimensional function.

Note:

While this is designed to accept a pointer to a const member function, the choice of whether the class pointer given in the template type `tclass` is const or not is up to the user.

Definition at line 308 of file `funct.h`.

Public Member Functions

- `funct_cmfptr` (tclass *tp, int(tclass::*fp)(double x, double &y, param_t &pa) const)
Specify the member function pointer.
- virtual int `operator()` (double x, double &y, param_t &pa)
Compute the function at point x, with result y.
- virtual double `operator()` (double x, param_t &pa)
Compute the function at point x, returning the result.

Protected Member Functions

- `funct_cmfptr` (const `funct_cmfptr` &f)
Copy constructor.
- `funct_cmfptr` & `operator=` (const `funct_cmfptr` &f)
Copy constructor.

Protected Attributes

- int(tclass::* `fptr`)(double x, double &y, param_t &pa) const
Storage for the const member function pointer.
- tclass * `tptr`
Store the pointer to the class instance.

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

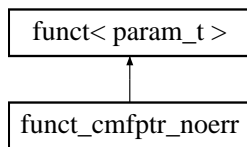
- `funct.h`

8.86 `funct_cmfp_ptr_noerr` Class Template Reference

Const member function pointer to a one-dimensional function returning the function value.

```
#include <funct.h>
```

Inheritance diagram for `funct_cmfp_ptr_noerr`:



8.86.1 Detailed Description

```
template<class tclass, class param_t> class funct_cmfp_ptr_noerr< tclass, param_t >
```

Const member function pointer to a one-dimensional function returning the function value.

Note:

While this is designed to accept a pointer to a const member function, the choice of whether the class pointer given in the template type `tclass` is const or not is up to the user.

Definition at line 440 of file `funct.h`.

Public Member Functions

- `funct_cmfp_ptr_noerr` (`tclass *tp`, `double(tclass::*fp)(double x, param_t &pa) const`)
Specify the member function pointer.
- virtual int `operator()` (`double x`, `double &y`, `param_t &pa`)
Compute the function at point x , with result y .
- virtual double `operator()` (`double x`, `param_t &pa`)
Compute the function at point x , returning the result.

Protected Member Functions

- `funct_cmfp_ptr_noerr` (`const funct_cmfp_ptr_noerr &f`)
Copy constructor.
- `funct_cmfp_ptr_noerr & operator=` (`const funct_cmfp_ptr_noerr &f`)
Copy constructor.

Protected Attributes

- `double(tclass::* fptr)` (`double x`, `param_t &pa`) const
Storage for the const member function pointer.
- `tclass * tptr`
Store the pointer to the class instance.

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

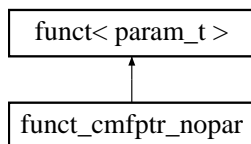
- `funct.h`

8.87 `funct_cmfp_ptr_nopar` Class Template Reference

Const member function pointer to a one-dimensional function with no parameters.

```
#include <funct.h>
```

Inheritance diagram for `funct_cmfp_ptr_nopar`:



8.87.1 Detailed Description

```
template<class tclass, class param_t> class funct_cmfp_ptr_nopar< tclass, param_t >
```

Const member function pointer to a one-dimensional function with no parameters.

Note:

While this is designed to accept a pointer to a const member function, the choice of whether the class pointer given in the template type `tclass` is const or not is up to the user.

Definition at line 571 of file `funct.h`.

Public Member Functions

- `funct_cmfp_ptr_nopar` (`tclass *tp`, `double(tclass::*fp)(double x) const`)
Specify the member function pointer.
- virtual int `operator()` (`double x`, `double &y`, `param_t &pa`)
Compute the function at point x , with result y .
- virtual double `operator()` (`double x`, `param_t &pa`)
Compute the function at point x , returning the result.

Protected Member Functions

- `funct_cmfp_ptr_nopar` (`const funct_cmfp_ptr_nopar &f`)
Copy constructor.
- `funct_cmfp_ptr_nopar & operator=` (`const funct_cmfp_ptr_nopar &f`)
Copy constructor.

Protected Attributes

- `double(tclass::* fptr)(double x) const`
Storage for the const member function pointer.
- `tclass * tptr`
Store the pointer to the class instance.

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

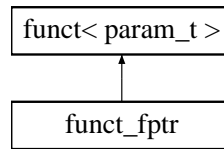
- `funct.h`

8.88 `funct_fptr` Class Template Reference

Function pointer to a function.

```
#include <funct.h>
```

Inheritance diagram for `funct_fptr`:



8.88.1 Detailed Description

```
template<class param_t> class funct_fptr< param_t >
```

Function pointer to a function.

Definition at line 71 of file `funct.h`.

Public Member Functions

- `funct_fptr` (`int(*fp)(double x, double &y, param_t &pa)`)
Specify the function pointer.
- virtual `int operator()` (`double x, double &y, param_t &pa`)
Compute the function at point `x`, with result `y`.
- virtual `double operator()` (`double x, param_t &pa`)
Compute the function at point `x`, returning the result.

Protected Member Functions

- `funct_fptr` (`const funct_fptr &f`)
Copy constructor.
- `funct_fptr & operator=` (`const funct_fptr &f`)
Copy constructor.

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:

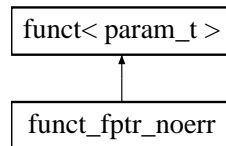
- `funct.h`

8.89 `funct_fptr_noerr` Class Template Reference

Function pointer to a function returning the function value.

```
#include <funct.h>
```

Inheritance diagram for `funct_fptr_noerr`:



8.89.1 Detailed Description

template<class param_t> class funct_fptr_noerr< param_t >

Function pointer to a function returning the function value.

Definition at line 127 of file funct.h.

Public Member Functions

- [funct_fptr_noerr](#) (double(*fp)(double x, param_t &pa))
Specify the function pointer.
- virtual int [operator\(\)](#) (double x, double &y, param_t &pa)
Compute the function at point x, with result y.
- virtual double [operator\(\)](#) (double x, param_t &pa)
Compute the function at point x, returning the result.

Protected Member Functions

- [funct_fptr_noerr](#) (const [funct_fptr_noerr](#) &f)
Copy constructor.
- [funct_fptr_noerr](#) & [operator=](#) (const [funct_fptr_noerr](#) &f)
Copy constructor.

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:

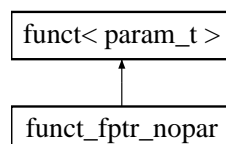
- funct.h

8.90 funct_fptr_nopar Class Template Reference

Function pointer to a function with no parameters.

```
#include <funct.h>
```

Inheritance diagram for funct_fptr_nopar::



8.90.1 Detailed Description

`template<class param_t> class funct_fptr_nopar< param_t >`

Function pointer to a function with no parameters.

Definition at line 183 of file `funct.h`.

Public Member Functions

- `funct_fptr_nopar` (`double(*fp)(double x)`)
Specify the function pointer.
- virtual `int operator()` (`double x, double &y, param_t &pa`)
Compute the function at point x , with result y .
- virtual `double operator()` (`double x, param_t &pa`)
Compute the function at point x , returning the result.

Protected Member Functions

- `funct_fptr_nopar` (`const funct_fptr_nopar &f`)
Copy constructor.
- `funct_fptr_nopar & operator=` (`const funct_fptr_nopar &f`)
Copy constructor.

Protected Attributes

- `double(* fptr)(double x)`
Storage for the function pointer.

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

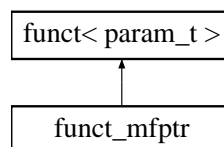
- `funct.h`

8.91 `funct_mfptr` Class Template Reference

Member function pointer to a one-dimensional function.

`#include <funct.h>`

Inheritance diagram for `funct_mfptr`:



8.91.1 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 240 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 int `operator()` (double x, double &y, param_t &pa)
Compute the function at point x, with result y.
- virtual double `operator()` (double x, param_t &pa)
Compute the function at point x, returning the result.

Protected Member Functions

- `funct_mfptr` (const `funct_mfptr` &f)
Copy constructor.
- `funct_mfptr & operator=` (const `funct_mfptr` &f)
Copy constructor.

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:

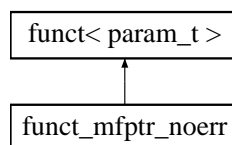
- `funct.h`

8.92 `funct_mfptr_noerr` Class Template Reference

Member function pointer to a one-dimensional function returning the function value.

```
#include <funct.h>
```

Inheritance diagram for `funct_mfptr_noerr`:



8.92.1 Detailed Description

```
template<class tclass, class param_t> class funct_mfptr_noerr< tclass, param_t >
```

Member function pointer to a one-dimensional function returning the function value.

Definition at line 371 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 int `operator()` (double x, double &y, param_t &pa)
Compute the function at point x, with result y.
- virtual double `operator()` (double x, param_t &pa)
Compute the function at point x, returning the result.

Protected Member Functions

- `funct_mfptr_noerr` (const `funct_mfptr_noerr` &f)
Copy constructor.
- `funct_mfptr_noerr` & `operator=` (const `funct_mfptr_noerr` &f)
Copy constructor.

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:

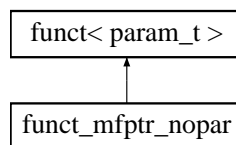
- `funct.h`

8.93 `funct_mfptr_nopar` Class Template Reference

Member function pointer to a one-dimensional function with no parameters.

```
#include <funct.h>
```

Inheritance diagram for `funct_mfptr_nopar`:



8.93.1 Detailed Description

```
template<class tclass, class param_t> class funct_mfptr_nopar< tclass, param_t >
```

Member function pointer to a one-dimensional function with no parameters.

Definition at line 504 of file `funct.h`.

Public Member Functions

- `funct_mfptr_nopar` (tclass *tp, double(tclass::*fp)(double x))
Specify the member function pointer.
- virtual int `operator()` (double x, double &y, param_t &pa)
Compute the function at point x, with result y.
- virtual double `operator()` (double x, param_t &pa)
Compute the function at point x, returning the result.

Protected Member Functions

- `funct_mfptr_nopar` (const `funct_mfptr_nopar` &f)
Copy constructor.
- `funct_mfptr_nopar & operator=` (const `funct_mfptr_nopar` &f)
Copy constructor.

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`

8.94 gaussian_2d Class Template Reference

Generate two random numbers from a normal distribution.

```
#include <gaussian_2d.h>
```

8.94.1 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 37 of file `gaussian_2d.h`.

Public Member Functions

- void `random` (double sigma, double &x, double &y)
Generate two numbers x and y from a distribution with zero mean and standard deviation sigma.

Data Fields

- `rng_t r`
The base random number generator.

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

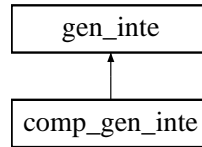
- `gaussian_2d.h`

8.95 gen_inte Class Template Reference

Generalized multi-dimensional integration [abstract base].

```
#include <gen_inte.h>
```

Inheritance diagram for gen_inte::



8.95.1 Detailed Description

```
template<class param_t, class func_t, class lfunc_t, class ufunc_t, class vec_t = ovector_base> class gen_inte< param_t,
func_t, lfunc_t, ufunc_t, vec_t >
```

Generalized multi-dimensional integration [abstract base].

Perform the generalized multi-dimensional integral:

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

The functions a_i and b_i are specified in the arguments `a` and `b` to the function [ginteg\(\)](#) or [ginteg_err\(\)](#).

In order to allow the user to specify only three functions (for the integrand, the lower limits, and the upper limits) the first argument to the limit and integrand functions is used to distinguish among the limits for each separate integral. So first argument to `a` for $a_0()$ is 0, and the first argument to `a` for $a_1()$ is 1, etc., and similarly for the upper limits specified in `b` and the integrands specified in `func`.

At present, the only implementation of this abstract base is in [comp_gen_inte](#).

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

Public Member Functions

- virtual double [ginteg](#) (func_t &func, size_t ndim, lfunc_t &a, ufunc_t &b, param_t &pa)=0
Integrate function `func` from $x_i = a_i(x_i)$ to $x_i = b_i(x_i)$ for $0 < i < \text{ndim} - 1$.
- virtual 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 $x_i = a_i(x_i)$ to $x_i = b_i(x_i)$ 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.
- bool [err_nonconv](#)
If true, call the error handler if the routine does not "converge".

Protected Attributes

- double `interror`
The uncertainty for the last integration computation.

8.95.2 Member Function Documentation

8.95.2.1 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 109 of file `gen_inte.h`.

8.95.2.2 virtual double `ginteg(func_t &func, size_t ndim, lfunc_t &a, ufunc_t &b, param_t &pa)` [pure virtual]

Integrate function `func` from $x_i = a_i(x_i)$ to $x_i = b_i(x_i)$ for $0 < i < \text{ndim} - 1$.

8.95.2.3 virtual int `ginteg_err(func_t &func, size_t ndim, lfunc_t &a, ufunc_t &b, param_t &pa, double &res, double &err)` [inline, virtual]

Integrate function `func` from $x_i = a_i(x_i)$ to $x_i = b_i(x_i)$ for $0 < i < \text{ndim} - 1$.

Definition at line 96 of file `gen_inte.h`.

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

- `gen_inte.h`

8.96 gen_test_number Class Template Reference

Generate number sequence for testing.

```
#include <misc.h>
```

8.96.1 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.

This class is used to generate combinations of coefficients for testing the polynomial solvers.

For example, the first 15 numbers generated by an object of type `gen_test_number<10>` are:

```
0  -1.000000e+00
1  -9.975274e-01
2  -8.807971e-01
3  -1.192029e-01
4  -2.472623e-03
5  +0.000000e+00
6  +2.472623e-03
7  +1.192029e-01
8  +8.807971e-01
```



```
9 +1.000000e+00
10 -1.000000e+00
11 -9.975274e-01
12 -8.807971e-01
13 -1.192029e-01
14 -2.472623e-03
```

Idea for future

Document what happens if tot is small

Definition at line 217 of file misc.h.

Public Member Functions

- 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 () , equal to tot divided by 20.

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

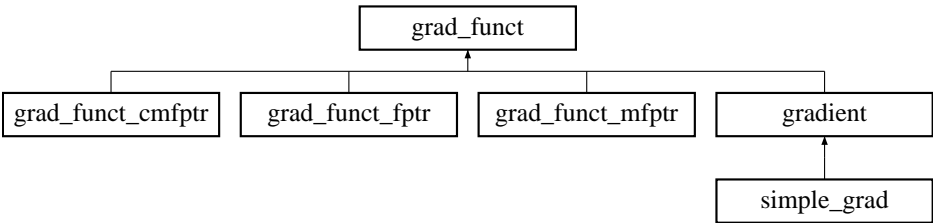
- misc.h

8.97 grad_func Class Template Reference

Gradient function [abstract base].

```
#include <multi_min.h>
```

Inheritance diagram for grad_func::



8.97.1 Detailed Description

```
template<class param_t, class vec_t = ovector_base> class grad_func< param_t, vec_t >
```

Gradient function [abstract base].

Default template arguments

- param_t - (no default)

- `vec_t` - [ovector_base](#)

Definition at line 43 of file `multi_min.h`.

Public Member Functions

- virtual int [operator\(\)](#) (size_t nv, vec_t &x, vec_t &g, param_t &pa)=0
Compute the [gradient](#) g at the point x .

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

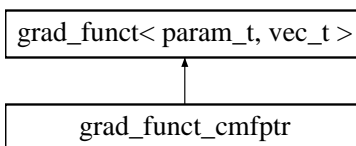
- `multi_min.h`

8.98 grad_funct_cmfpnr Class Template Reference

Const member function pointer to a [gradient](#).

```
#include <multi_min.h>
```

Inheritance diagram for `grad_funct_cmfpnr`:



8.98.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base> class grad_funct_cmfpnr< tclass, param_t, vec_t >

Const member function pointer to a [gradient](#).

Default template arguments

- `tclass` - (no default)
- `param_t` - (no default)
- `vec_t` - [ovector_base](#)

Definition at line 158 of file `multi_min.h`.

Public Member Functions

- [grad_funct_cmfpnr](#) (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &g, param_t &pa) const)
Specify the member function pointer.
- 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) const`
Member function pointer.
- `tclass * tptr`
Class pointer.

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

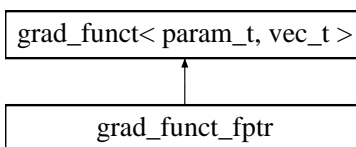
- `multi_min.h`

8.99 grad_funct_fptr Class Template Reference

Function pointer to a [gradient](#).

```
#include <multi_min.h>
```

Inheritance diagram for `grad_funct_fptr`:

**8.99.1 Detailed Description**

```
template<class param_t, class vec_t = ovector_base> class grad_funct_fptr< param_t, vec_t >
```

Function pointer to a [gradient](#).

Default template arguments

- `param_t` - (no default)
- `vec_t` - [ovector_base](#)

Definition at line 61 of file `multi_min.h`.

Public Member Functions

- `grad_funct_fptr(int(*fp)(size_t nv, vec_t &x, vec_t &g, param_t &pa))`
Specify the function pointer.
- `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

- `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:

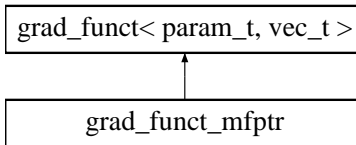
- `multi_min.h`

8.100 grad_funct_mfptr Class Template Reference

Member function pointer to a [gradient](#).

```
#include <multi_min.h>
```

Inheritance diagram for grad_funct_mfptr::



8.100.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_base> class grad_funct_mfptr< tclass, param_t, vec_t >
```

Member function pointer to a [gradient](#).

Default template arguments

- `tclass` - (no default)
- `param_t` - (no default)
- `vec_t` - [ovector_base](#)

Definition at line 107 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 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:

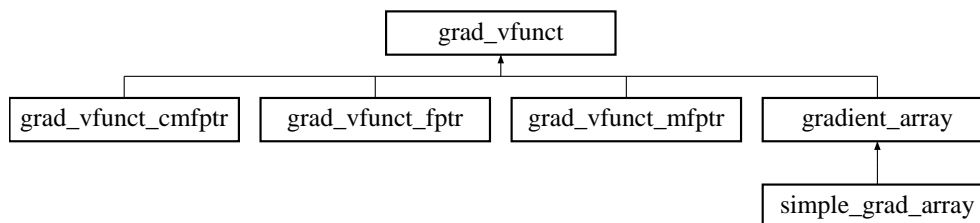
- multi_min.h

8.101 grad_vfunct Class Template Reference

Base class for a [gradient](#) function using arrays [abstract base].

```
#include <multi_min.h>
```

Inheritance diagram for grad_vfunct::



8.101.1 Detailed Description

template<class param_t, size_t nv> class grad_vfunct< param_t, nv >

Base class for a [gradient](#) function using arrays [abstract base].

Definition at line 284 of file multi_min.h.

Public Member Functions

- virtual int [operator\(\)](#) (size_t nvar, double x[nv], double g[nv], param_t &pa)=0
Compute the [gradient](#) g at the point x .

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

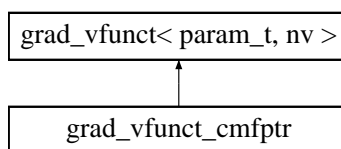
- multi_min.h

8.102 grad_vfunct_cmfpnr Class Template Reference

Const member function pointer to a [gradient](#).

```
#include <multi_min.h>
```

Inheritance diagram for grad_vfunct_cmfpnr::



8.102.1 Detailed Description

template<class tclass, class param_t, size_t nv> class grad_vfunct_cmfpnr< tclass, param_t, nv >

Const member function pointer to a [gradient](#).

Definition at line 393 of file multi_min.h.

Public Member Functions

- [grad_vfunct_cmfpnr](#) (tclass *tp, int(tclass::*fp)(size_t nvar, double x[nv], double g[nv], param_t &pa) const)
Specify the member function pointer.
- 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

- `int(tclass::* fptr)(size_t nvar, double x[nv], double g[nv], param_t &pa) const`
Member function pointer.
- `tclass * tptr`
Class pointer.

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

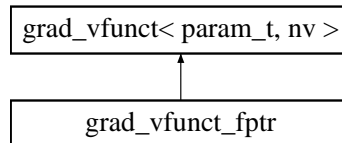
- `multi_min.h`

8.103 grad_vfunct_fptr Class Template Reference

Function pointer to a [gradient](#).

```
#include <multi_min.h>
```

Inheritance diagram for `grad_vfunct_fptr`:

**8.103.1 Detailed Description**

```
template<class param_t, size_t nv> class grad_vfunct_fptr< param_t, nv >
```

Function pointer to a [gradient](#).

Note:

`nv` should be greater than zero

Definition at line 303 of file `multi_min.h`.

Public Member Functions

- `grad_vfunct_fptr (int(*fp)(size_t nvar, double x[nv], double g[nv], param_t &pa))`
Specify the member function pointer.
- `virtual int operator\(\) (size_t nvar, double x[nv], double g[nv], param_t &pa)`
Compute the [gradient](#) ∇ at the point 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:

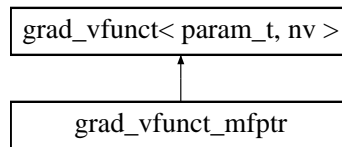
- `multi_min.h`

8.104 grad_vfunct_mfptr Class Template Reference

Member function pointer to a [gradient](#).

```
#include <multi_min.h>
```

Inheritance diagram for grad_vfunct_mfptr::



8.104.1 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 346 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 int [operator\(\)](#) (size_t nvar, double x[nv], double g[nv], param_t &pa)
Compute the [gradient](#) g at the point 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:

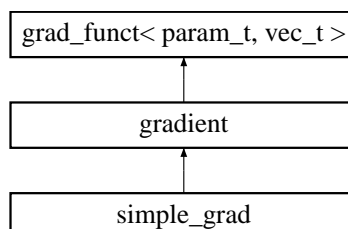
- multi_min.h

8.105 gradient Class Template Reference

Class for automatically computing gradients [abstract base].

```
#include <multi_min.h>
```

Inheritance diagram for gradient::



8.105.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_base> class gradient< param_t, func_t, vec_t >

Class for automatically computing gradients [abstract base].

Default template arguments

- param_t - (no default)
- func_t - (no default)
- vec_t - [ovector_base](#)

Definition at line 209 of file multi_min.h.

Public Member Functions

- 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)=0
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:

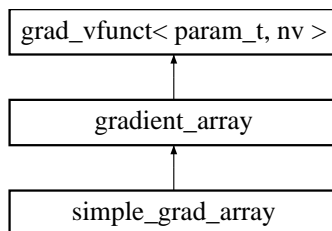
- multi_min.h

8.106 gradient_array Class Template Reference

Base class for automatically computing gradients with arrays [abstract base].

```
#include <multi_min.h>
```

Inheritance diagram for gradient_array::



8.106.1 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 [abstract base].

Definition at line 444 of file multi_min.h.

Public Member Functions

- 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)=0
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:

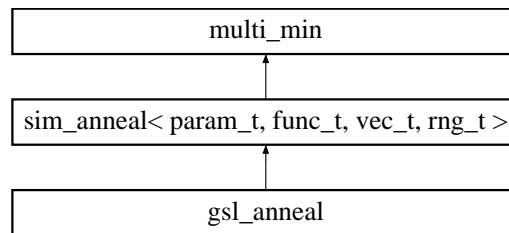
- multi_min.h

8.107 gsl_anneal Class Template Reference

Multidimensional minimization by simulated annealing (GSL).

```
#include <gsl_anneal.h>
```

Inheritance diagram for gsl_anneal::



8.107.1 Detailed Description

```
template<class param_t, class func_t = multi_func<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc, class rng_t = gsl_rng> class gsl_anneal< param_t, func_t, vec_t, alloc_vec_t, alloc_t, rng_t >
```

Multidimensional minimization by simulated annealing (GSL).

This class is a modification of simulated annealing as implemented in GSL in the function `gsl_siman_solve()`. It acts as a generic multidimensional minimizer for any function given a generic temperature schedule specified by the user.

The simulated annealing algorithm proposes a displacement of one coordinate of the previous point by

$$x_{i,\text{new}} = \text{step_size}_i(2u_i - 1) + x_{i,\text{old}}$$

where the u_i are random numbers between 0 and 1. The displacement is accepted or rejected based on the Metropolis method. The random number generator is set in the parent, [sim_anneal](#).

There are a large variety of strategies for choosing the temperature evolution. To offer the user the largest possible flexibility, the temperature evolution is controlled by the virtual functions [start\(\)](#) and [next\(\)](#) which can be freely changed by creating descendant.

The default behavior is as follows: Initially, the step sizes are chosen to be 10 and the temperature to be 1. Each iteration decreases the temperature by a factor of 1.5 for each step, and the minimizer is finished when the next decrease would bring the temperature below [multi_min::tolx](#). If none of the [multi_min::ntrial](#) steps in a particular iteration changes the value of the minimum, and the step sizes are greater than 100 times [multi_min::tolx](#), then the step sizes are decreased by a factor of 1.5 for the next iteration.

If `multi_min::verbose` is greater than zero, then `mmin()` will print out information and/or request a keypress after the function iterations for each temperature.

An example demonstrating the usage of this class is given in `examples/ex_anneal.cpp` and in the [Simulated annealing example](#).

See also a multi-threaded version of this class in [anneal_mt](#).

Idea for future

There's `x0`, `old_x`, `new_x`, `best_x`, and `x`? There's probably some duplication here which could be avoided.

Idea for future

- Implement a more general simulated annealing routine which would allow the solution of discrete problems like the Traveling Salesman problem.

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

Public Member Functions

- 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").
- template<class vec2_t >
int `set_step` (size_t nv, vec2_t &stepv)
Set the step sizes.

Data Fields

- double `boltz`
Boltzmann factor (default 1.0).

Protected Member Functions

- virtual int `next` (size_t nvar, vec_t &x_old, double min_old, vec_t &x_new, double min_new, double &T, size_t n_moves, bool &finished)
Determine how to change the minimization for the next iteration.
- virtual int `start` (size_t nvar, double &T)
Setup initial temperature and stepsize.
- virtual int `allocate` (size_t n, double boltz_factor=1.0)
Allocate memory for a minimizer over n dimensions with stepsize step and Boltzmann factor boltz_factor.
- virtual int `free` ()
Free allocated memory.
- virtual int `step` (vec_t &sx, int nvar)
Make a step to a new attempted minimum.

Protected Attributes

- alloc_t `ao`
Allocation object.
- ovector `step_vec`
Vector of step sizes.

Storage for present, next, and best vectors

- `alloc_vec_t x`
- `alloc_vec_t new_x`
- `alloc_vec_t best_x`
- `alloc_vec_t old_x`

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

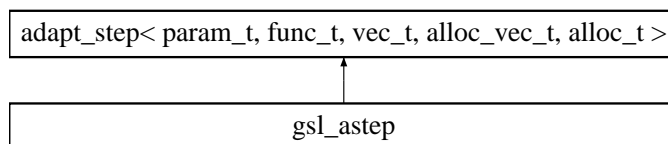
- `gsl_anneal.h`

8.108 gsl_astep Class Template Reference

Adaptive ODE stepper (GSL).

```
#include <gsl_astep.h>
```

Inheritance diagram for `gsl_astep`:



8.108.1 Detailed Description

template<class `param_t`, class `func_t` = `ode_func_t`<`param_t`>, class `vec_t` = `ovector_base`, 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 ODE stepper (GSL).

To modify the ODE stepper which is used, use the [adapt_step::set_step\(\)](#).

Default template arguments

- `param_t` - [multi_func_t](#)
- `func_t` - [ode_func_t](#)
- `vec_t` - [ovector_base](#)
- `alloc_vec_t` - [ovector](#)
- `alloc_t` - [ovector_alloc](#)

Definition at line 244 of file `gsl_astep.h`.

Public Member Functions

- virtual int [astep_derivs](#) (double &`x`, double &`h`, double `xmax`, size_t `n`, `vec_t` &`y`, `vec_t` &`dydx`, `vec_t` &`yerr`, `param_t` &`pa`, `func_t` &`derivs`)
Make an adaptive integration step of the system `derivs` with derivatives.
- virtual int [astep](#) (double &`x`, double &`h`, double `xmax`, size_t `n`, `vec_t` &`y`, `vec_t` &`dydx_out`, `vec_t` &`yerr`, `param_t` &`pa`, `func_t` &`derivs`)
Make an adaptive integration step of the system `derivs`.

Data Fields

- [gsl_ode_control](#)< [vec_t](#) > [con](#)
Control specification.

Protected Member Functions

- [int evolve_apply](#) (double &t, double &h, double t1, size_t nvar, [vec_t](#) &y, [vec_t](#) &dydx, [vec_t](#) &yout2, [vec_t](#) &yerr, [vec_t](#) &dydx_out2, [param_t](#) &pa, [func_t](#) &derivs)
Apply the evolution for the next adaptive step.

Protected Attributes

- [alloc_t](#) [ao](#)
Memory allocator for objects of type [alloc_vec_t](#).
- [alloc_vec_t](#) [yout](#)
Temporary storage for yout.
- [alloc_vec_t](#) [dydx_int](#)
Internal storage for dydx.
- [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.
- [size_t](#) [msize](#)
The size of the allocated vectors.

8.108.2 Member Function Documentation

8.108.2.1 [virtual int astep](#) (double &x, double &h, double xmax, size_t n, [vec_t](#) &y, [vec_t](#) &dydx_out, [vec_t](#) &yerr, [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, `dydx_out` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

If the system `derivs` or the base stepper return a non-zero value, the adaptive step is aborted and `y` is unmodified. The error handler is never called.

Implements [adapt_step](#).

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

8.108.2.2 [virtual int astep_derivs](#) (double &x, double &h, double xmax, size_t n, [vec_t](#) &y, [vec_t](#) &dydx, [vec_t](#) &yerr, [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, `dydx` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

If the base stepper returns a non-zero value, the step is aborted and `y` is unmodified. The error handler is never called.

Implements [adapt_step](#).

Definition at line 376 of file `gsl_astepp.h`.

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

- `gsl_astepp.h`

8.109 gsl_bsimp Class Template Reference

Bulirsch-Stoer implicit ODE stepper (GSL).

```
#include <gsl_bsimp.h>
```

8.109.1 Detailed Description

```
template<class param_t, class func_t, class jac_func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t =
ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc> class gsl_bsimp<
param_t, func_t, jac_func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t >
```

Bulirsch-Stoer implicit ODE stepper (GSL).

Bader-Deuffhard implicit extrapolative stepper ([Bader83](#)).

Note:

The variable `h_next` was defined in the original GSL version has been removed here, as it was unused by the stepper routine.

Things to document

More detailed documentation

Todo

Ensure error handling is sensible if the "derivs" or [jacobian](#) functions return a non-zero value

Todo

Create an example with a stiff diff eq. which requires this kind of stepper

Idea for future

I don't like setting `yerr` to `GSL_POSINF`, there should be a better way to force an adaptive stepper which is calling this stepper to readjust the stepsize.

Idea for future

Some of these functions can be moved out of this header file

Idea for future

Rework internal arrays as uvectors?

Idea for future

The function [step_local\(\)](#) is actually its own ODE stepper and could be reimplemented as an object of type [odestep](#)

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

Public Member Functions

- int [reset](#) ()
Reset 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, jac_func_t &jac)
Perform an integration step.

Protected Member Functions

- int [compute_weights](#) (const double y[], double w[], size_t n)
Compute weights.
- size_t [deuf_kchoice](#) (double eps2, size_t dimension)
Calculate a choice for the "order" of the method, using the Deuffhard criteria.
- int [poly_extrap](#) (gsl_matrix *dloc, const double x[], const unsigned int i_step, const double x_i, const vec_t &y_i, vec_t &y_0, vec_t &y_0_err, double work[])
Polynomial extrapolation.
- int [step_local](#) (const double t0, const double h_total, const unsigned int n_step, const double y[], const double yp_loc[], const vec_t &dfdt_loc, const mat_t &dfdy_loc, vec_t &y_out)
Basic implicit Bulirsch-Stoer step.
- int [allocate](#) (size_t n)
Allocate memory for a system of size n.
- void [free](#) ()
Free allocated memory.

Protected Attributes

- size_t [dim](#)
Size of allocated vectors.
- alloc_t [ao](#)
Memory allocator for objects of type alloc_vec_t.
- mat_alloc_t [mo](#)
Memory allocator for objects of type alloc_mat_t.
- func_t * [funcp](#)
Function specifying derivatives.
- jac_func_t * [jfuncp](#)
Jacobian.
- param_t * [pap](#)
User-specified parameter.
- gsl_matrix * [d](#)
Workspace for extrapolation.
- gsl_matrix * [a_mat](#)
Workspace for linear system matrix.
- double [ex_wk](#) [[sequence_max](#)]
Workspace for extrapolation.
- size_t [order](#)
Order of last step.

State info

- size_t [k_current](#)
- size_t [k_choice](#)
- double [eps](#)

Workspace for extrapolation step

- double * [yp](#)

- double * **y_save**
- double * **yerr_save**
- double * **y_extrap_save**
- alloc_vec_t **y_extrap_sequence**
- double * **extrap_work**
- alloc_vec_t **dfdt**
- alloc_vec_t **y_temp**
- alloc_vec_t **delta_temp**
- double * **weight**
- alloc_mat_t **dfdy**

Workspace for the basic stepper

- alloc_vec_t **rhs_temp**
- double * **delta**

Static Protected Attributes

- static const int **sequence_count** = 8
Desc.
- static const int **sequence_max** = 7
Desc.

8.109.2 Member Function Documentation

8.109.2.1 size_t deuf_kchoice (double eps2, size_t dimension) [inline, protected]

Calculate a choice for the "order" of the method, using the Deufhard criteria.

Used in the [allocate\(\)](#) function.

Definition at line 178 of file `gsl_bsimp.h`.

8.109.2.2 int poly_extrap (gsl_matrix * dloc, const double x[], const unsigned int i_step, const double x_i, const vec_t & y_i, vec_t & y_0, vec_t & y_0_err, double work[]) [inline, protected]

Polynomial extrapolation.

Compute the step of index `i_step` using polynomial extrapolation to evaluate functions by fitting a polynomial to estimates (`x_i`, `y_i`) and output the result to `y_0` and `y_0_err`.

The index `i_step` begins with zero.

Definition at line 234 of file `gsl_bsimp.h`.

8.109.2.3 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, jac_func_t & jac) [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.

Definition at line 498 of file `gsl_bsimp.h`.

8.109.2.4 `int step_local (const double t0, const double h_total, const unsigned int n_step, const double y[], const double yp_loc[], const vec_t & dfdt_loc, const mat_t & dfdy_loc, vec_t & y_out)` `[inline, protected]`

Basic implicit Bulirsch-Stoer step.

Divide the step `h_total` into `n_step` smaller steps and do the Bader-Deuflhard semi-implicit iteration. This function starts at `t0` with function values `y`, derivatives `yp_loc`, and information from the Jacobian to compute the final value `y_out`.

Definition at line 284 of file `gsl_bsimp.h`.

8.109.3 Field Documentation

8.109.3.1 `double ex_wk[sequence_max]` `[protected]`

Workspace for extrapolation.

(This state variable was named 'x' in GSL.)

Definition at line 126 of file `gsl_bsimp.h`.

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

- `gsl_bsimp.h`

8.110 gsl_chebapp Class Reference

Chebyshev approximation (GSL).

```
#include <gsl_chebapp.h>
```

8.110.1 Detailed Description

Chebyshev approximation (GSL).

Approximate a function on a finite interval 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` (const `gsl_chebapp` &gc)
Copy constructor.
- `gsl_chebapp` & `operator=` (const `gsl_chebapp` &gc)
Copy constructor.
- `template<class param_t, class func_t >`
`int init` (func_t &func, size_t ord, double a1, double b1, param_t &vp)
Initialize a Chebyshev approximation of the function func over the interval from a1 to b1.
- `template<class vec_t >`
`int init` (double a1, double b1, size_t ord, vec_t &v)
Create an approximation from a vector of coefficients.
- `template<class vec_t >`
`int init_func_values` (double a1, double b1, size_t ord, vec_t &fval)
Create an approximation from a vector of function values.
- `double eval` (double x) const
Evaluate the approximation.

- double `eval_n` (size_t n, double x) const
Evaluate the approximation to a specified order.
- int `eval_err` (double x, double &result, double &abserr)
Evaluate the approximation and give the uncertainty.
- int `eval_n_err` (size_t n, double x, double &result, double &abserr)
Evaluate the approximation to a specified order and give the uncertainty.
- double `get_coefficient` (size_t ix) const
Get a coefficient.
- int `set_coefficient` (size_t ix, double co)
Set a coefficient.
- int `get_endpoints` (double &la, double &lb)
Return the endpoints of the approximation.
- template<class vec_t >
int `get_coefficients` (size_t n, vec_t &v) const
Get the coefficients.
- template<class vec_t >
int `set_coefficients` (size_t n, vec_t &v)
Set the coefficients.
- int `deriv` (gsl_chebapp &gc) const
Make gc an approximation to the derivative.
- int `integ` (gsl_chebapp &gc) const
Make gc an approximation to the integral.

Protected Attributes

- `uvector c`
Coefficients.
- size_t `order`
Order of the approximation.
- double `a`
Lower end of the interval.
- double `b`
Upper end of the interval.
- size_t `order_sp`
Single precision order.
- `uvector f`
Function evaluated at Chebyshev points.
- bool `init_called`
True if init has been called.

8.110.2 Member Function Documentation

8.110.2.1 double get_coefficient (size_t ix) const [inline]

Get a coefficient.

Legal values of the argument are 0 to `order` (inclusive)

Definition at line 310 of file `gsl_chebapp.h`.

8.110.2.2 int init (func_t &func, size_t ord, double a1, double b1, param_t &vp) [inline]

Initialize a Chebyshev approximation of the function `func` over the interval from `a1` to `b1`.

The interval must be specified so that $a < b$, so `a` and `b` are swapped if this is not the case.

Definition at line 105 of file `gsl_chebapp.h`.

8.110.2.3 int set_coefficient (size_t ix, double co) [inline]

Set a coefficient.

Legal values of the argument are 0 to `order` (inclusive)

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

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

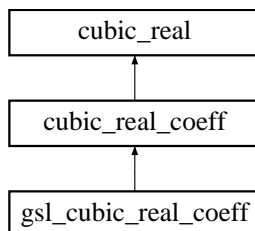
- `gsl_chebapp.h`

8.111 gsl_cubic_real_coeff Class Reference

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

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

Inheritance diagram for `gsl_cubic_real_coeff`:

**8.111.1 Detailed Description**

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

Definition at line 494 of file `poly.h`.

Public Member Functions

- 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").
- 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()`.

8.111.2 Member Function Documentation**8.111.2.1 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()`.

This is an alternative to the function `gsl_poly_complex_solve_cubic()` with some small corrections to ensure finite values for some cubics. See `src/other/poly_ts.cpp` for more.

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

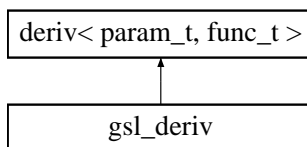
- [poly.h](#)

8.112 gsl_deriv Class Template Reference

Numerical differentiation (GSL).

```
#include <gsl_deriv.h>
```

Inheritance diagram for gsl_deriv::



8.112.1 Detailed Description

```
template<class param_t, class func_t = funct<param_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`.

The derivative computation will never fail, but the results will be incorrect for sufficiently difficult functions or if the step size is not properly chosen.

Some successive derivative computations can be made more efficient by using the optimized stepsize in `gsl_deriv::h_opt`, which is set by the most recent last derivative computation.

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.

An example demonstrating the usage of this class is given in `examples/ex_deriv.cpp` and the [Numerical differentiation example](#).

Idea for future

Include the forward and backward GSL derivatives

Definition at line 88 of file `gsl_deriv.h`.

Public Member Functions

- 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 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.

8.112.2 Member Function Documentation

8.112.2.1 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 232 of file `gsl_deriv.h`.

8.112.3 Field Documentation

8.112.3.1 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 106 of file `gsl_deriv.h`.

8.112.3.2 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 114 of file `gsl_deriv.h`.

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

- `gsl_deriv.h`

8.113 gsl_fft Class Reference

Real mixed-radix fast Fourier transform.

```
#include <gsl_fft.h>
```

8.113.1 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:

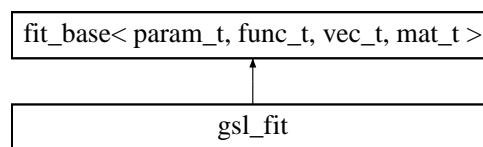
- [gsl_fft.h](#)

8.114 gsl_fit Class Template Reference

Non-linear least-squares fitting class (GSL).

```
#include <gsl_fit.h>
```

Inheritance diagram for `gsl_fit`:



8.114.1 Detailed Description

```
template<class param_t, class func_t = fit_func<param_t>, class vec_t = ovector_base, class mat_t = omatrix_base, 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_base

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 87 of file gsl_fit.h.

Data Structures

- struct [func_par](#)
A structure for passing to the functions [func\(\)](#), [dfunc\(\)](#), and [fdfunc\(\)](#).

Public Member Functions

- 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](#).

8.114.2 Member Function Documentation

8.114.2.1 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 χ^2 is returned in `chi2`.

Implements [fit_base](#).

Definition at line 271 of file `gsl_fit.h`.

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

- `gsl_fit.h`

8.115 gsl_fit::func_par Struct Reference

A structure for passing to the functions [func\(\)](#), [dfunc\(\)](#), and [fdfunc\(\)](#).

```
#include <gsl_fit.h>
```

8.115.1 Detailed Description

```
template<class param_t, class func_t = fit_func<param_t>, class vec_t = ovector_base, class mat_t = omatrix_base, 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 402 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:

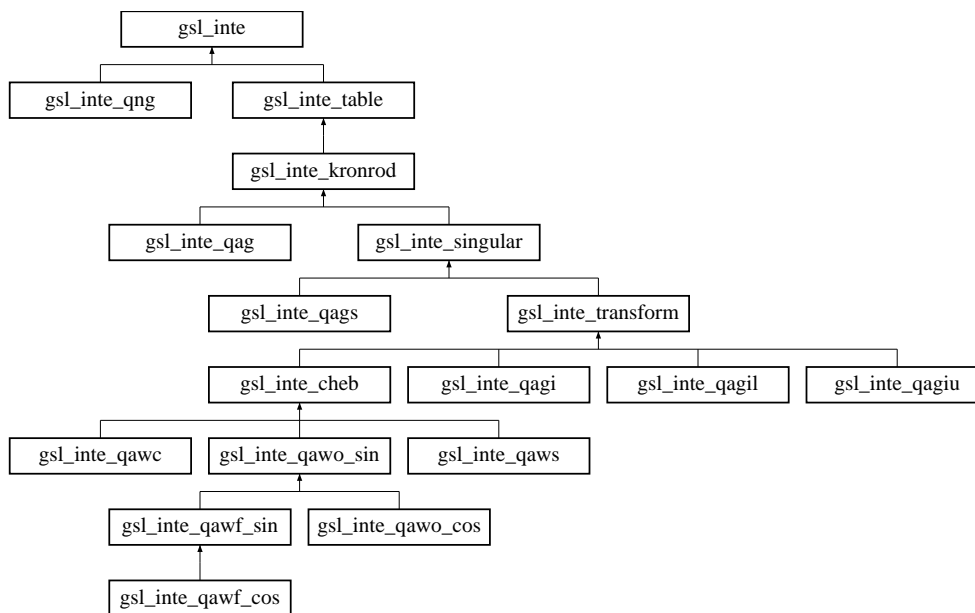
- `gsl_fit.h`

8.116 gsl_inte Class Reference

GSL integration base.

```
#include <gsl_inte.h>
```

Inheritance diagram for `gsl_inte::`:



8.116.1 Detailed Description

GSL integration base.

This base class does not perform any actual integration, but just provides functions to be used in the integration classes based on GSL.

Definition at line 57 of file `gsl_inte.h`.

Protected Member Functions

- `double rescale_error` (double err, const double result_abs, const double result_asc)
Rescale errors appropriately.

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

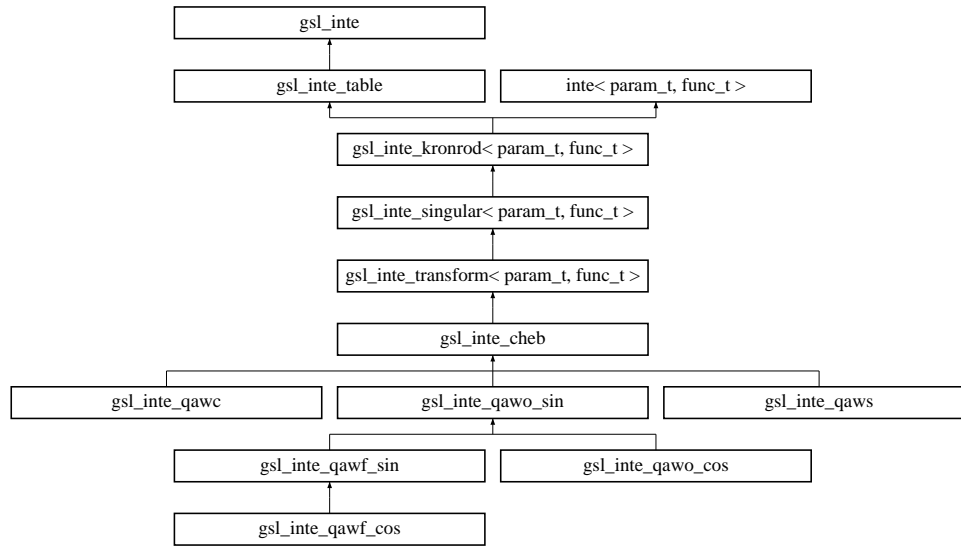
- `gsl_inte.h`

8.117 gsl_inte_cheb Class Template Reference

Chebyshev integration base class (GSL).


```
#include <gsl_inte_qawc.h>
```

Inheritance diagram for `gsl_inte_cheb`:



8.117.1 Detailed Description

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

Chebyshev integration base class (GSL).

This class provides the basic Chebyshev integration functions for use in the GSL-based integration classes which require them.

Definition at line 41 of file `gsl_inte_qawc.h`.

Public Member Functions

- void `compute_moments` (double cc, double *moment)
Compute the Chebyshev moments.
- void `gsl_integration_qcheb` (func_t &f, double a, double b, double *cheb12, double *cheb24, param_t &pa)
Perform the integration.

8.117.2 Member Function Documentation

8.117.2.1 void `gsl_integration_qcheb` (func_t &f, double a, double b, double *cheb12, double *cheb24, param_t &pa)
[inline]

Perform the integration.

piessens,robert,appl. math. & progr. div. - k.u.leuven de doncker,elise,appl. math. & progr. div. - k.u.leuven

this routine computes the chebyshev series expansion of degrees 12 and 24 of a function using a fast fourier transform method $f(x) = \sum_{k=1, \dots, 13} (\text{cheb12}(k) * t(k-1, x))$, $f(x) = \sum_{k=1, \dots, 25} (\text{cheb24}(k) * t(k-1, x))$, where $t(k, x)$ is the chebyshev polynomial of degree k .

x - double precision vector of dimension 11 containing the values $\cos(k * \pi / 24)$, $k = 1, \dots, 11$

fval - double precision vector of dimension 25 containing the function values at the points $(b+a+(b-a)*\cos(k*\pi/24))/2$, $k = 0, \dots, 24$, where (a,b) is the approximation interval. fval(1) and fval(25) are divided by two (these values are destroyed at output).

on return cheb12 - double precision vector of dimension 13 containing the chebyshev coefficients for degree 12

cheb24 - double precision vector of dimension 25 containing the chebyshev coefficients for degree 24

Definition at line 108 of file gsl_inte_qawc.h.

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

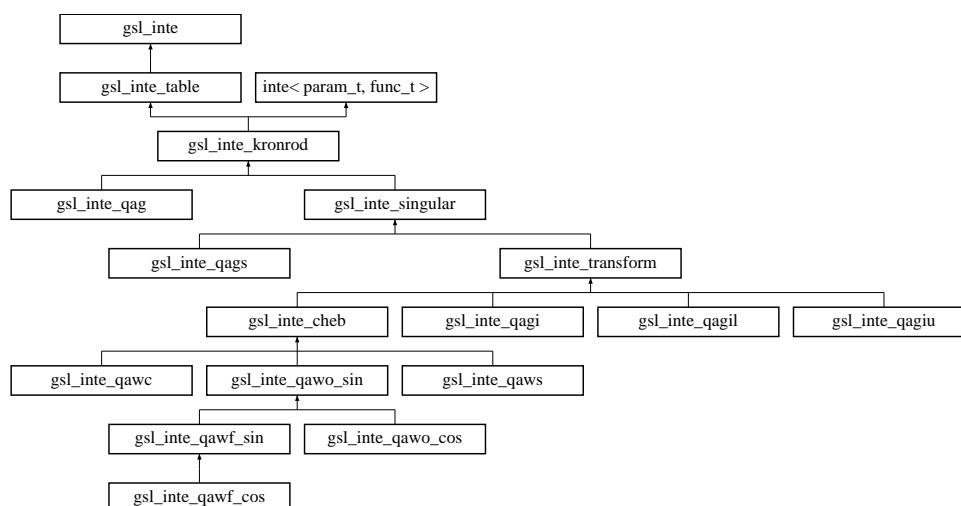
- gsl_inte_qawc.h

8.118 gsl_inte_kronrod Class Template Reference

Basic Gauss-Kronrod integration class (GSL).

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for gsl_inte_kronrod::



8.118.1 Detailed Description

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

Basic Gauss-Kronrod integration class (GSL).

This class provides the basic Gauss-Kronrod integration function for some of the GSL-based integration classes. The casual end-user should use the classes explained in the [Integration](#) section of the User's guide.

Definition at line 615 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 GSL Gauss-Kronrod integration function.

8.118.2 Member Function Documentation

8.118.2.1 `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 GSL Gauss-Kronrod integration function.

Given abscissas and weights, this performs the integration of `func` between `a` and `b`, providing a result with uncertainties.

This function is designed for use with the values given in the `o2scl_inte_qag_coeffs` namespace.

This function never calls the error handler.

Idea for future

This function, in principle, could be replaced with a generic integration pointer.

Reimplemented in `gsl_inte_transform`.

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

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

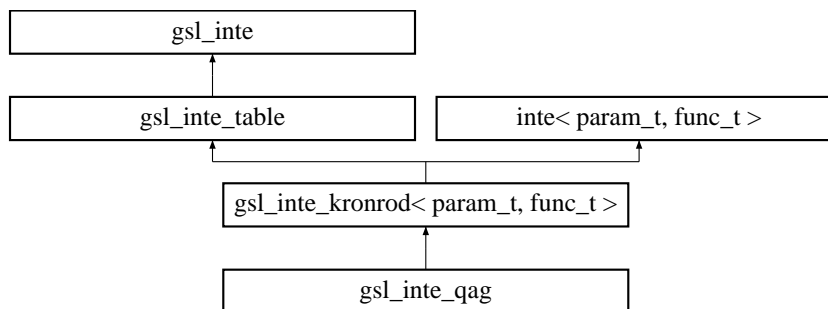
- `gsl_inte_qag_b.h`

8.119 gsl_inte_qag Class Template Reference

Adaptive integration a function with finite limits of integration (GSL).

`#include <gsl_inte_qag.h>`

Inheritance diagram for `gsl_inte_qag::`



8.119.1 Detailed Description

`template<class param_t, class func_t = funct<param_t>> class gsl_inte_qag< param_t, func_t >`

Adaptive integration a function with finite limits of integration (GSL).

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using `gsl_inte_table::set_workspace()`.

There are a few possible normal errors:

- Iteration limit exceeds workspace in `gsl_inte_qag::qag()` . - `gsl_einval`
- Tolerance cannot be achieved with given value of 'tolx' and 'tolf' in `gsl_inte_qag::qag()` . - `gsl_ebadtol`

- Cannot reach tolerance because of roundoff error on first attempt in `gsl_inte_qag::qag()` . - `gsl_eround`
- Could not integrate function in `gsl_inte_qag::qag()` (it may have returned a non-finite result) . - `gsl_efailed`

There are also a few convergence errors which will not be called if `inte::err_nonconv` is false (the default is true)

- A maximum of 1 iteration was insufficient in `gsl_inte_qag::qag()` . - `gsl_emaxiter`
- Bad integrand behavior in `gsl_inte_qag::qag()` . - `gsl_esign`
- Maximum number of subdivisions 'value' reached in `gsl_inte_qag::qag()` . - `gsl_emaxiter`

Todo

Verbose output has been setup for this class, but this needs to be done for some of the other GSL-like integrators

Things to document

Document use of `last_iter`

Definition at line 85 of file `gsl_inte_qag.h`.

Public Member Functions

- `gsl_inte_qag` (int key=1)
Create an integrator with the specified key.
- int `set_key` (int key)
Set the number of integration points.
- int `get_key` ()
Return the current value of the key being used 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.
- const char * `type` ()
Return string denoting type ("gsl_inte_qag").

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.

Protected Attributes

- int `lkey`
Select the number of integration points.

8.119.2 Member Function Documentation

8.119.2.1 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 123 of file `gsl_inte_qag.h`.

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

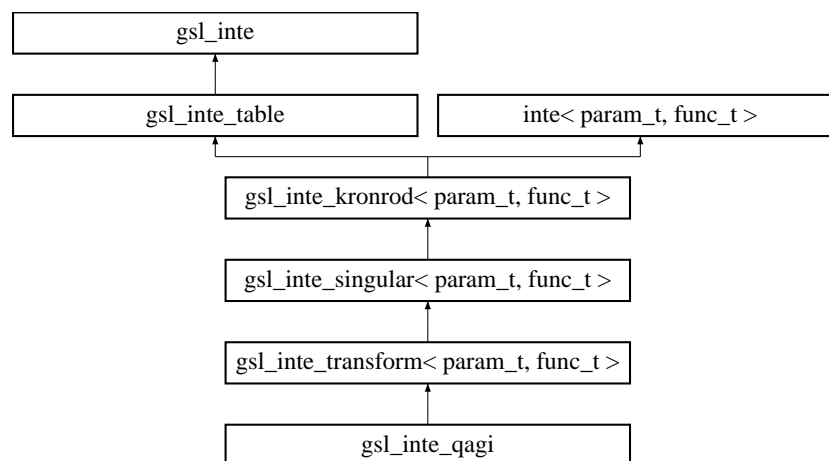
- `gsl_inte_qag.h`

8.120 gsl_inte_qagi Class Template Reference

Integrate a function from $-\infty$ to ∞ (GSL).

```
#include <gsl_inte_qagi.h>
```

Inheritance diagram for `gsl_inte_qagi`:



8.120.1 Detailed Description

```
template<class param_t, class func_t = funct<param_t>> class gsl_inte_qagi< param_t, func_t >
```

Integrate a function from $-\infty$ to ∞ (GSL).

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using `gsl_inte_table::set_workspace()`.

Definition at line 42 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 ∞ .
- virtual int `integ_err` (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
Integrate function func from ∞ to ∞ giving result res and error err.

Protected Member Functions

- virtual double `transform` (func_t &func, double t, param_t &pa)
Transformation to $t \in (0, 1]$.

8.120.2 Member Function Documentation

8.120.2.1 virtual double integ (func_t &func, double a, double b, param_t &pa) [inline, virtual]

Integrate function `func` from $-\infty$ to ∞ .

The values given in `a` and `b` are ignored

Implements `inte`.

Definition at line 52 of file `gsl_inte_qagi.h`.

8.120.2.2 virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2) [inline, virtual]

Integrate function `func` from ∞ to ∞ giving result `res` and error `err`.

The values `a` and `b` are ignored

Implements `inte`.

Definition at line 64 of file `gsl_inte_qagi.h`.

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

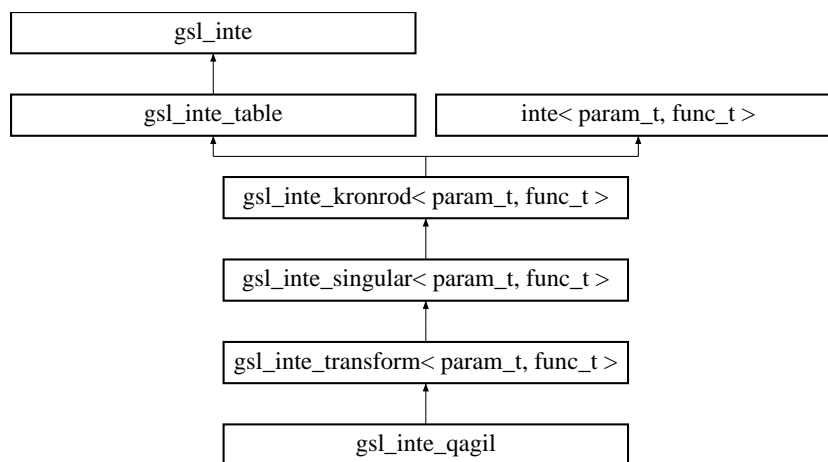
- `gsl_inte_qagi.h`

8.121 gsl_inte_qagil Class Template Reference

Integrate a function from $-\infty$ to `b` (GSL).

```
#include <gsl_inte_qagil.h>
```

Inheritance diagram for `gsl_inte_qagil`:



8.121.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class gsl_inte_qagil< param_t, func_t >

Integrate a function from $-\infty$ to b (GSL).

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using `gsl_inte_table::set_workspace()`.

Definition at line 42 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.

8.121.2 Member Function Documentation

8.121.2.1 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.

Implements `inte`.

Definition at line 61 of file `gsl_inte_qagil.h`.

8.121.2.2 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.

Implements [inte](#).

Definition at line 74 of file `gsl_inte_qagil.h`.

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

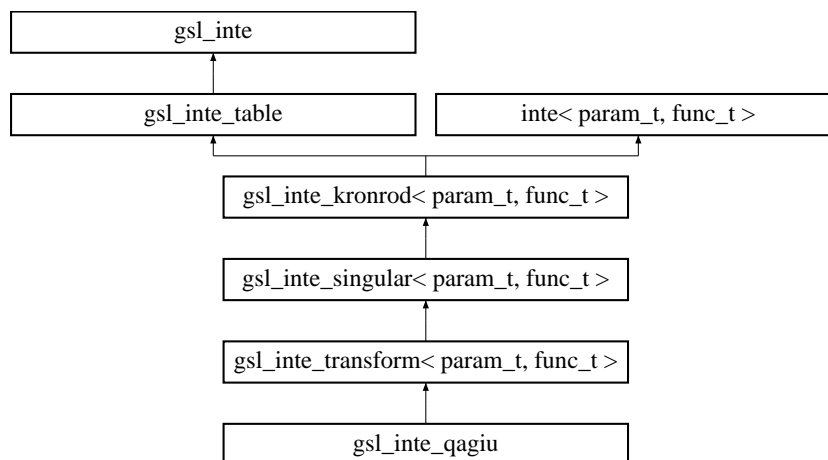
- `gsl_inte_qagil.h`

8.122 gsl_inte_qagiu Class Template Reference

Integrate a function from a to ∞ (GSL).

```
#include <gsl_inte_qagiu.h>
```

Inheritance diagram for `gsl_inte_qagiu::`



8.122.1 Detailed Description

```
template<class param_t, class func_t = funct<param_t>> class gsl_inte_qagiu< param_t, func_t >
```

Integrate a function from a to ∞ (GSL).

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using [gsl_inte_table::set_workspace\(\)](#) .

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 49 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 ∞ .
- virtual int [integ_err](#) (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
Integrate function func from a to ∞ 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.

8.122.2 Member Function Documentation

8.122.2.1 virtual double integ (func_t &func, double a, double b, param_t &pa) [inline, virtual]

Integrate function func from a to ∞ .

The value b is ignored.

Implements [inte](#).

Definition at line 69 of file gsl_inte_qagiu.h.

8.122.2.2 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 ∞ giving result res and error err.

The value b is ignored.

Implements [inte](#).

Definition at line 82 of file gsl_inte_qagiu.h.

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

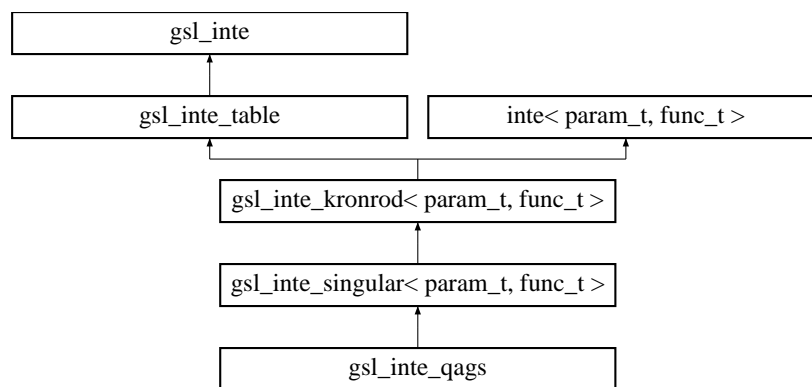
- gsl_inte_qagiu.h

8.123 gsl_inte_qags Class Template Reference

Integrate a function with a singularity (GSL).

```
#include <gsl_inte_qags.h>
```

Inheritance diagram for gsl_inte_qags::



8.123.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class gsl_inte_qags< param_t, func_t >

Integrate a function with a singularity (GSL).

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using [gsl_inte_table::set_workspace\(\)](#) .

Todo

The convergence errors need to be handled correctly in function qags in [gsl_inte_qag_b.h](#).

Definition at line 45 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:

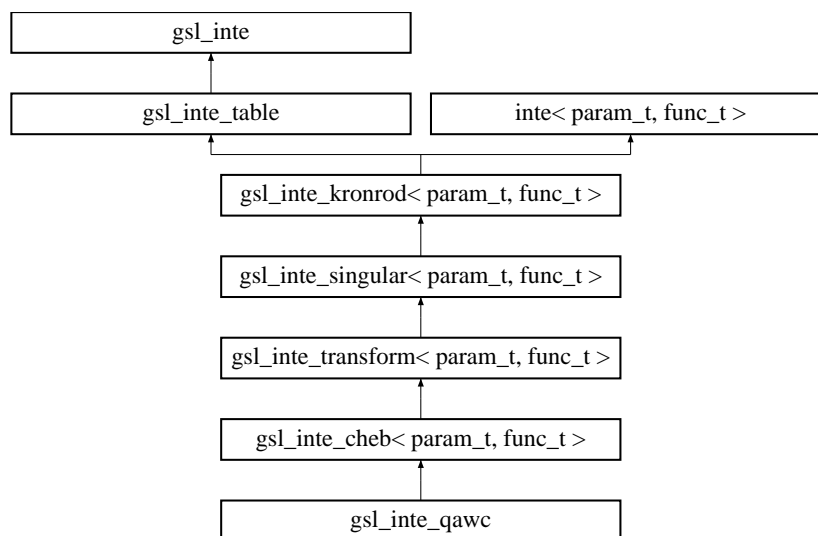
- [gsl_inte_qags.h](#)

8.124 gsl_inte_qawc Class Template Reference

Adaptive Cauchy principal value integration (GSL).

```
#include <gsl_inte_qawc.h>
```

Inheritance diagram for [gsl_inte_qawc](#)::



8.124.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_qawc< param_t, func_t >

Adaptive Cauchy principal value integration (GSL).

The location of the singularity must be specified before-hand in [gsl_inte_qawc::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 not be specified in the argument `func` to [integ\(\)](#). This is different from how the [cern_cauchy](#) operates.

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using [gsl_inte_table::set_workspace\(\)](#).

Idea for future

Make [cern_cauchy](#) and this class consistent in the way which they require the user to provide the denominator in the integrand

Definition at line 332 of file `gsl_inte_qawc.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.

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)
The full GSL integration routine called by integ_err().

- void [qc25c](#) (func_t &func, double a, double b, double c, double *result, double *abserr, int *err_reliable, param_t &pa)
25-point quadrature for Cauchy principal values
- virtual double [transform](#) (func_t &func, double x, param_t &pa)
Add the singularity to the function.
- const char * [type](#) ()
Return string denoting type ("gsl_inte_qawc").

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

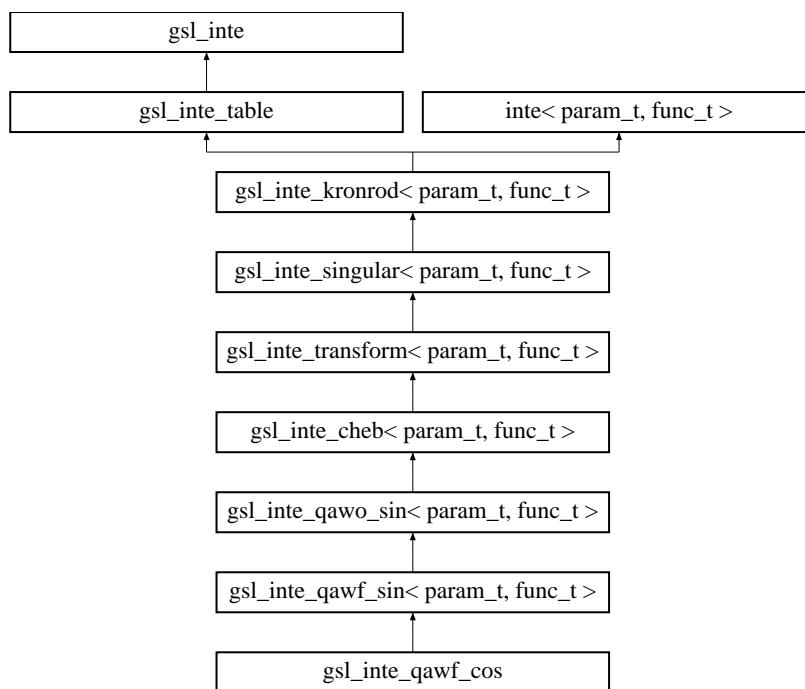
- `gsl_inte_qawc.h`

8.125 gsl_inte_qawf_cos Class Template Reference

Adaptive integration a function with finite limits of integration (GSL).

```
#include <gsl_inte_qawf.h>
```

Inheritance diagram for `gsl_inte_qawf_cos`:



8.125.1 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).

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using [gsl_inte_table::set_workspace\(\)](#) .

Todo

Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Definition at line 337 of file `gsl_inte_qawf.h`.

Public Member Functions

- 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)
Add the oscillating part to the integrand.
- const char * [type](#) ()
Return string denoting type ("gsl_inte_qawf_cos").

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

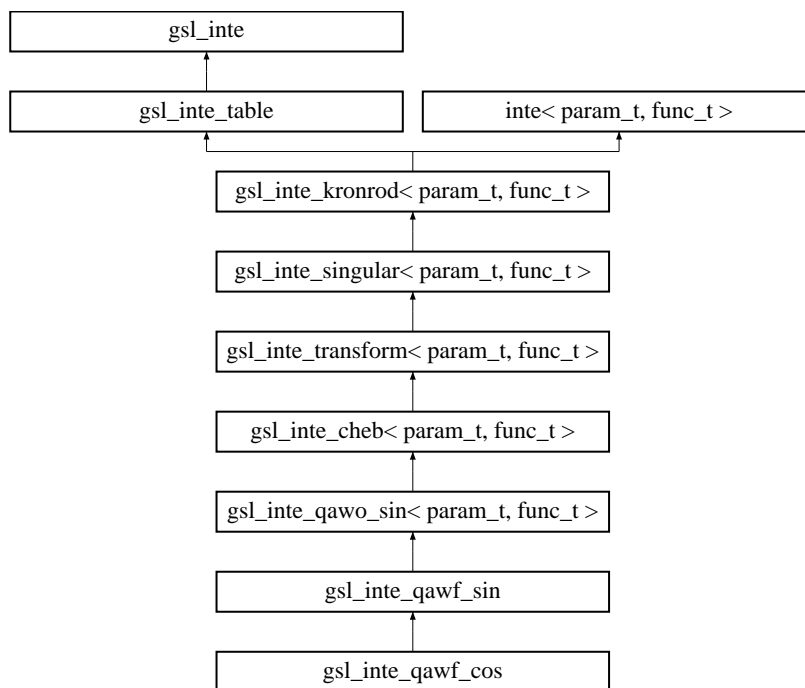
- gsl_inte_qawf.h

8.126 gsl_inte_qawf_sin Class Template Reference

Adaptive integration for oscillatory integrals (GSL).

```
#include <gsl_inte_qawf.h>
```

Inheritance diagram for gsl_inte_qawf_sin::



8.126.1 Detailed Description

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

Adaptive integration for oscillatory integrals (GSL).

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using [gsl_inte_table::set_workspace\(\)](#) .

Todo

Improve documentation a little

Definition at line 44 of file gsl_inte_qawf.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`.

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)
The full GSL integration routine called by [integ_err\(\)](#).
- virtual double [transform](#) (func_t &func, double x, param_t &pa)
Add the oscillating part to the integrand.
- const char * [type](#) ()
Return string denoting type ("`gsl_inte_qawf_sin`").

Protected Attributes

- gsl_integration_workspace * [cyclew](#)
The integration workspace.

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

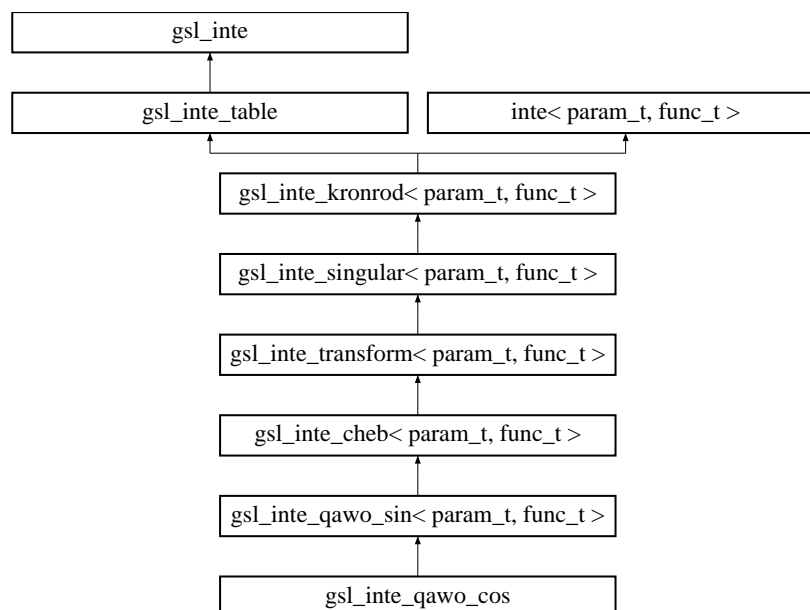
- gsl_inte_qawf.h

8.127 gsl_inte_qawo_cos Class Template Reference

Adaptive integration a function with finite limits of integration (GSL).

```
#include <gsl_inte_qawo.h>
```

Inheritance diagram for `gsl_inte_qawo_cos::`



8.127.1 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).

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using [gsl_inte_table::set_workspace\(\)](#) .

Todo

Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Definition at line 659 of file `gsl_inte_qawo.h`.

Public Member Functions

- 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)
Add the oscillating part to the integrand.
- const char * [type](#) ()
Return string denoting type ("`gsl_inte_qawo_cos`").

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

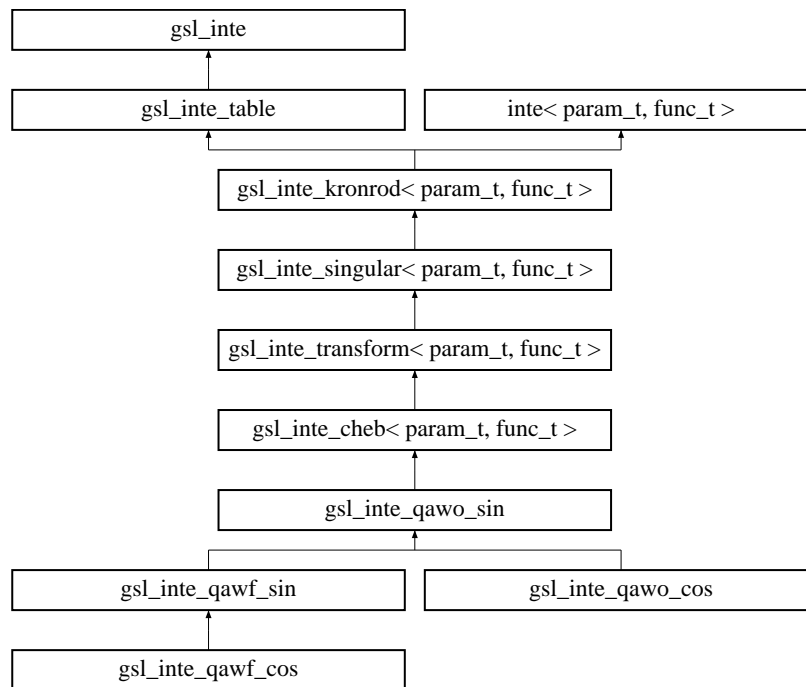
- `gsl_inte_qawo.h`

8.128 gsl_inte_qawo_sin Class Template Reference

Adaptive integration for oscillatory integrals (GSL).

```
#include <gsl_inte_qawo.h>
```

Inheritance diagram for `gsl_inte_qawo_sin`:



8.128.1 Detailed Description

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

Adaptive integration for oscillatory integrals (GSL).

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using `gsl_inte_table::set_workspace()`.

Todo

Improve documentation

Definition at line 43 of file `gsl_inte_qawo.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.

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)
The full GSL integration routine called by [integ_err\(\)](#).
- 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)
25-point quadrature for oscillating functions
- virtual double [transform](#) (func_t &func, double x, param_t &pa)
Add the oscillating part to the integrand.
- const char * [type](#) ()
Return string denoting type ("gsl_inte_qawo_sin").

Protected Attributes

- gsl_integration_qawo_table * [otable](#)
The integration workspace.

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

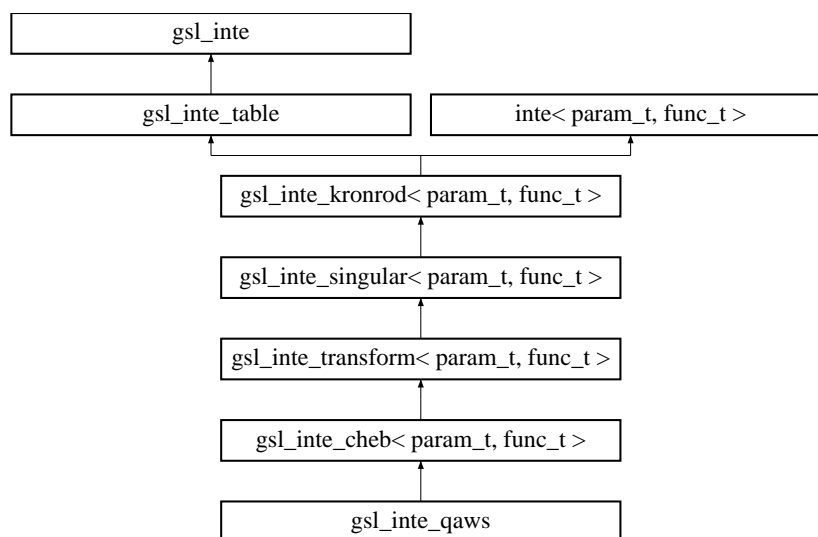
- [gsl_inte_qawo.h](#)

8.129 gsl_inte_qaws Class Template Reference

QAWS integration (GSL).

```
#include <gsl_inte_qaws.h>
```

Inheritance diagram for `gsl_inte_qaws::`



8.129.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_qaws< param_t, func_t >

QAWS integration (GSL).

The number of subdivisions of the original interval which this class is allowed to make is dictated by the workspace size for the integration class, which can be set using `gsl_inte_table::set_workspace()` .

Note:

This is unfinished.

Todo

Finish this!

Definition at line 45 of file `gsl_inte_qaws.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.

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.
- double `fn_qaws` (double t, int params)
- double `fn_qaws_L` (double x, int params)
- double `fn_qaws_R` (double x, int params)
- void `compute_result` (const double *r, const double *cheb12, const double *cheb24, double *result12, double *result24)
- 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)
- 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)
- double `fn_qaws` (double x, int params)
- double `fn_qaws_L` (double x, int params)
- double `fn_qaws_R` (double x, int params)
- void `compute_result` (const double *r, const double *cheb12, const double *cheb24, double *result12, double *result24)
- virtual double `transform` (func_t &func, double x, param_t &pa)
Desc.
- const char * `type` ()
Return string denoting type ("gsl_inte_qaws").

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

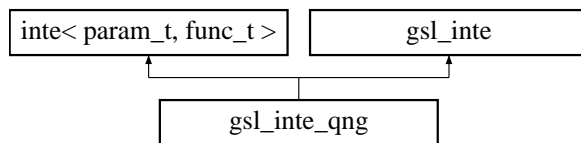
- `gsl_inte_qaws.h`

8.130 gsl_inte_qng Class Template Reference

Non-adaptive integration from a to b (GSL).

```
#include <gsl_inte_qng.h>
```

Inheritance diagram for gsl_inte_qng::



8.130.1 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).

The function [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 [inte::tolx](#) and [inte::tolf](#).

The error handler is called if the 87-point integration fails to produce the desired accuracy. If [inte::err_nonconv](#) is false (the default is true), then the error handler is never called and when the desired accuracy is not obtained the result of the 87-point integration is returned along with the associated error.

The return value of the function to be integrated is ignored.

Todo

Shouldn't feval be last_iter ?

Idea for future

Compare directly with GSL as is done in `gsl_inte_qag_ts`.

Definition at line 237 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.

8.130.2 Member Function Documentation

8.130.2.1 `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 `b` giving result `res` and error `err`.

Idea for future

Allow user to change 0.5e28

Implements [inte](#).

Definition at line 267 of file `gsl_inte_qng.h`.

8.130.3 Field Documentation

8.130.3.1 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 252 of file `gsl_inte_qng.h`.

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

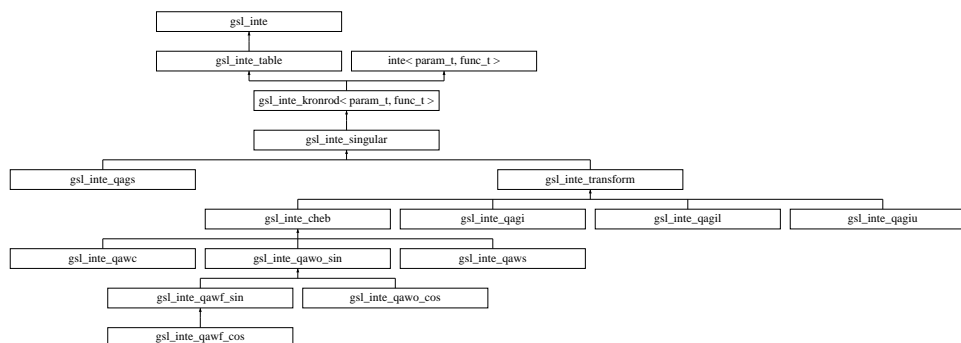
- `gsl_inte_qng.h`

8.131 gsl_inte_singular Class Template Reference

Base class for integrating a function with a singularity (GSL).

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for `gsl_inte_singular`:



8.131.1 Detailed Description

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

Base class for integrating a function with a singularity (GSL).

This class contains the extrapolation [table](#) mechanics and the base integration function for singular integrals from GSL. The casual end-user should use the classes explained in the [Integration](#) section of the User's guide.

Idea for future

Some of the functions inside this class could be moved out of header files?

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

Data Structures

- struct [extrapolation_table](#)
A structure for extrapolation for `gsl_inte_qags`.

Protected Member Functions

- void [initialise_table](#) (struct [extrapolation_table](#) *table)
Initialize the [table](#).
- void [append_table](#) (struct [extrapolation_table](#) *table, double y)
Append a result to the [table](#).
- int [test_positivity](#) (double result, double resabs)
Test a result for positivity.
- void [qelg](#) (struct [extrapolation_table](#) *table, double *result, double *abserr)
Determines the limit of a given sequence of approximations.
- int [large_interval](#) (gsl_integration_workspace *workspace)
Determine if an interval is large.
- void [reset_nrmx](#) (gsl_integration_workspace *workspace)
Reset workspace to work on the interval with the largest error.
- int [increase_nrmx](#) (gsl_integration_workspace *workspace)
Increase workspace.
- int [qags](#) (func_t &func, const int qn, const double xgk[], const double wg[], const double wkg[], 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)
Integration function.

8.131.2 Member Function Documentation

8.131.2.1 int [qags](#) (func_t &func, const int qn, const double xgk[], const double wg[], const double wkg[], 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) [inline, protected]

Integration function.

Idea for future

Remove goto statements?

Output iteration information

Definition at line 1024 of file `gsl_inte_qag_b.h`.

8.131.2.2 void [qelg](#) (struct [extrapolation_table](#) *table, double *result, double *abserr) [inline, protected]

Determines the limit of a given sequence of approximations.

This function determines the limit of a given sequence of approximations, by means of the epsilon algorithm of P. Wynn. an estimate of the absolute error is also given. the condensed epsilon [table](#) is computed. only those elements needed for the computation of the next diagonal are preserved.

Quadpack documentation

```

c
c      list of major variables
c      -----
c
c      e0      - the 4 elements on which the computation of a new
c      e1      element in the epsilon table is based
c      e2
c      e3
c
c              e0
c              e3  e1  new
c              e2
c      newelm - number of elements to be computed in the new
c              diagonal
c      error  - error = abs(e1-e0)+abs(e2-e1)+abs(new-e2)
c      result - the element in the new diagonal with least value
c              of error
c
c      machine dependent constants
c      -----
c
c      epmach is the largest relative spacing.
c      oflow is the largest positive magnitude.
c      limexp is the maximum number of elements the epsilon
c      table can contain. if this number is reached, the upper
c      diagonal of the epsilon table is deleted.
c

```

Definition at line 814 of file `gsl_inte_qag_b.h`.

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

- `gsl_inte_qag_b.h`

8.132 `gsl_inte_singular::extrapolation_table` Struct Reference

A structure for extrapolation for [gsl_inte_qags](#).

```
#include <gsl_inte_qag_b.h>
```

8.132.1 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](#).

Idea for future

Move this to a new class, with [qe1g\(\)](#) as a method

Definition at line 743 of file `gsl_inte_qag_b.h`.

Data Fields

- `size_t n`
Index of new element in the first column.

- double [rlist2](#) [52]
Lower diagonals of the triangular epsilon [table](#).
- size_t [nres](#)
Number of calls.
- double [res3la](#) [3]
Three most recent results.

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

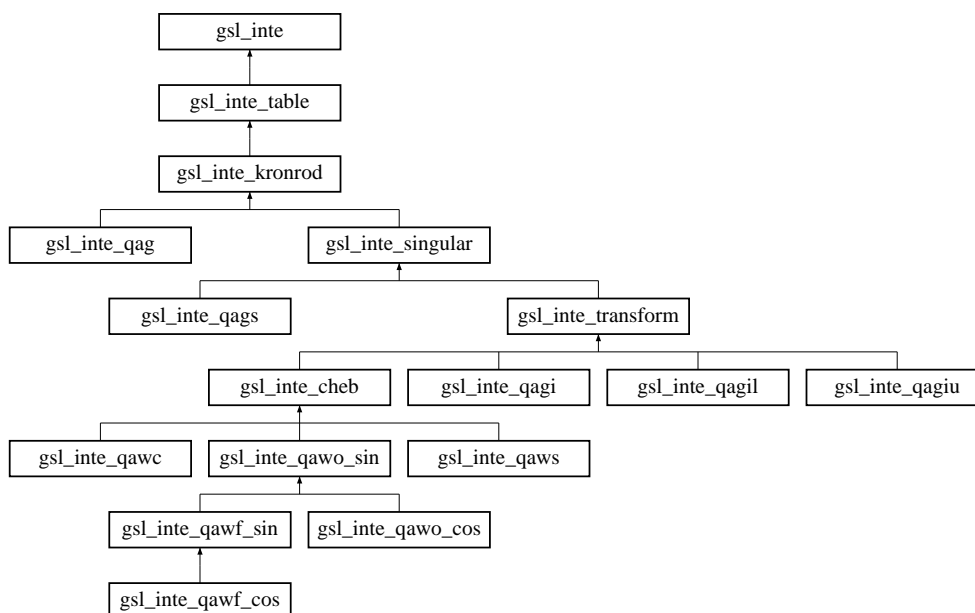
- `gsl_inte_qag_b.h`

8.133 gsl_inte_table Class Reference

Base routines for the GSL adaptive integration classes.

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for `gsl_inte_table`:



8.133.1 Detailed Description

Base routines for the GSL adaptive integration classes.

This class contains several functions for manipulating the GSL integration workspace. The casual end-user should use the classes explained in the [Integration](#) section of the User's guide.

Todo

Make the workspace size protected

Idea for future

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 `wkspc` though. Some work on this front is already in [gsl_inte_qag_b.h](#).

QUADPACK workspace documentation:

```

c    parameters (meaning at output)
c    limit - integer
c           maximum number of error estimates the list
c           can contain
c    last  - integer
c           number of error estimates currently in the list
c    maxerr - integer
c           maxerr points to the nrmax-th largest error
c           estimate currently in the list
c    ermax  - double precision
c           nrmax-th largest error estimate
c           ermax = elist(maxerr)
c    elist  - double precision
c           vector of dimension last containing
c           the error estimates
c    iord   - integer
c           vector of dimension last, the first k elements
c           of which contain pointers to the error
c           estimates, such that
c           elist(iord(1)), ..., elist(iord(k))
c           form a decreasing sequence, with
c           k = last if last.le.(limit/2+2), and
c           k = limit+1-last otherwise
c    nrmax  - integer
c    maxerr = iord(nrmax)

c    alist  - real
c           vector of dimension at least limit, the first
c           last elements of which are the left
c           end points of the subintervals in the partition
c           of the given integration range (a,b)
c    blist  - real
c           vector of dimension at least limit, the first
c           last elements of which are the right
c           end points of the subintervals in the partition
c           of the given integration range (a,b)
c    rlist  - real
c           vector of dimension at least limit, the first
c           last elements of which are the
c           integral approximations on the subintervals
c    elist  - real
c           vector of dimension at least limit, the first
c           last elements of which are the moduli of the
c           absolute error estimates on the subintervals
c    iord   - integer
c           vector of dimension at least limit, the first k
c           elements of which are pointers to the
c           error estimates over the subintervals,
c           such that elist(iord(1)), ...,
c           elist(iord(k)) form a decreasing sequence,
c           with k = last if last.le.(limit/2+2), and
c           k = limit+1-last otherwise
c    last   - integer
c           number of subintervals actually produced in the
c           subdivision process

```

Definition at line 532 of file `gsl_inte_qag_b.h`.

Public Member Functions

- `int set_wsplace (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 ith 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 (default 1000).

8.133.2 Member Function Documentation

8.133.2.1 void qpsrt (gsl_integration_workspace * workspace)

Sort the workspace.

This routine maintains the descending ordering in the list of the local error estimated resulting from the interval subdivision process. at each call two error estimates are inserted using the sequential search method, top-down for the largest error estimate and bottom-up for the smallest error estimate.

Originally written in QUADPACK by

```
piessens, robert, appl. math. & progr. div. - k.u.leuven
de doncker, elise, appl. math. & progr. div. - k.u.leuven
```

translated into C for GSL by Brian Gough, and then rewritten for O2scl.

8.133.2.2 void retrieve (const gsl_integration_workspace * workspace, double * a, double * b, double * r, double * e)

Retrieve the ith 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:

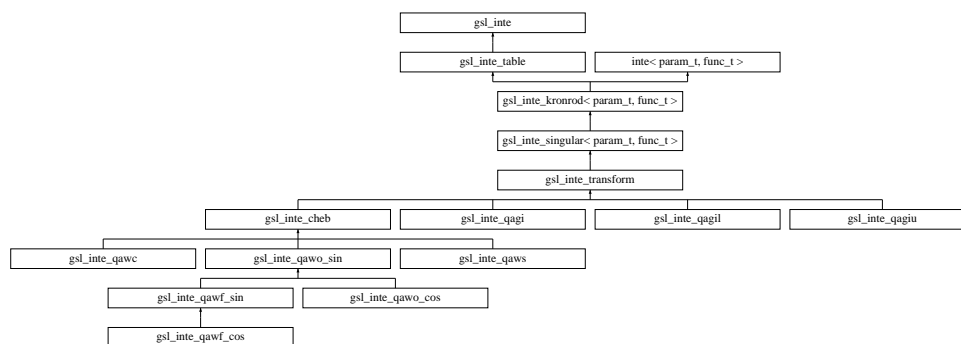
- gsl_inte_qag_b.h

8.134 gsl_inte_transform Class Template Reference

Integrate a function with a singularity (GSL) [abstract base].

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for `gsl_inte_transform`:



8.134.1 Detailed Description

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

Integrate a function with a singularity (GSL) [abstract base].

This class contains the GSL-based integration function for applying transformations to the user-defined integrand. The casual end-user should use the classes explained in the [Integration](#) section of the User's guide.

Definition at line 1414 of file `gsl_inte_qag_b.h`.

Public Member Functions

- virtual double [transform](#) (func_t &func, double t, param_t &pa)=0
The transformation to apply to the user-supplied function.
- 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.

8.134.2 Member Function Documentation

8.134.2.1 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 function 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.

This function never calls the error handler.

Reimplemented from [gsl_inte_kronrod](#).

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

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

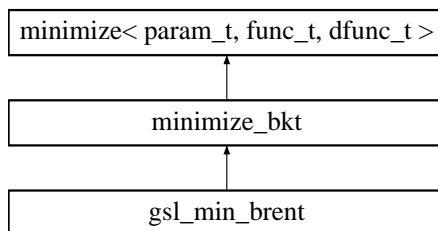
- `gsl_inte_qag_b.h`

8.135 gsl_min_brent Class Template Reference

One-dimensional minimization using Brent's method (GSL).

```
#include <gsl_min_brent.h>
```

Inheritance diagram for `gsl_min_brent`:



8.135.1 Detailed Description

```
template<class param_t, class func_t = funct<param_t>> class gsl_min_brent< param_t, func_t >
```

One-dimensional minimization using Brent's method (GSL).

The minimization in the function `min_bkt()` is complete when the bracketed interval is smaller than $\text{tol} = \text{tolx} + \text{tolf} \cdot \text{min}$, where $\text{min} = \min(|\text{lower}|, |\text{upper}|)$.

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.

Note:

There was a bug in this minimizer which was fixed for GSL-1.11 which has also been fixed here.

Definition at line 72 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 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, the initial bracketing interval, and the corresponding function values.
- int `iterate` ()
Perform an iteration.
- virtual int `min_bkt` (double &x2, double x1, double x3, double &fmin, param_t &pa, func_t &func)
Calculate the minimum fmin 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](#)

8.135.2 Member Function Documentation

8.135.2.1 `virtual int min_bkt (double & x2, double x1, double x3, double & fmin, param_t & pa, func_t & func)`
[inline, virtual]

Calculate the minimum `fmin` 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.

Implements [minimize_bkt](#).

Definition at line 323 of file `gsl_min_brent.h`.

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

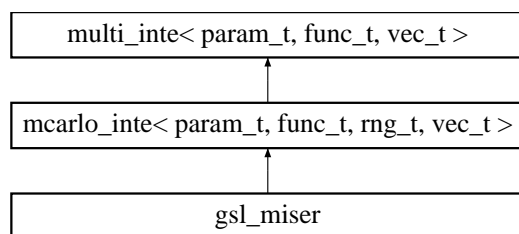
- `gsl_min_brent.h`

8.136 gsl_miser Class Template Reference

Multidimensional integration using Miser Monte Carlo (GSL).

```
#include <gsl_miser.h>
```

Inheritance diagram for `gsl_miser::`



8.136.1 Detailed Description

`template<class param_t, class func_t = multi_func<param_t>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_miser< param_t, func_t, rng_t, vec_t, alloc_vec_t, alloc_t >`

Multidimensional integration using Miser Monte Carlo (GSL).

Things to document

Document the fact that `min_calls` and `min_calls_per_bisection` need to be set beforehand

Things to document

Document the member data

Todo

The testing file calls the error handler on the `composite_inte` section. Fix this.

Based on [Press90](#).

Definition at line 72 of file `gsl_miser.h`.

Public Member Functions

- virtual int `allocate` (size_t ldim)
Allocate memory.
- virtual int `free` ()
Free allocated memory.
- virtual int `miser_minteg_err` (func_t &func, size_t ndim, const vec_t &xl, const vec_t &xu, size_t calls, param_t &pa, double &res, double &err)
Integrate function func over the hypercube from $x_i = xl_i$ to $x_i = xu_i$ for $0 < i < ndim-1$.
- 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 double `minteg` (func_t &func, size_t ndim, const vec_t &a, const vec_t &b, param_t &pa)
Integrate function func over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i < ndim-1$.
- virtual const char * `type` ()
Return string denoting type ("gsl_miser").

Data Fields

- double `dither`
Introduce random variation into bisection (default 0.0).
- double `estimate_frac`
Specify fraction of function calls for estimating variance.
- double `alpha`
How estimated variances for two sub-regions are combined.

- size_t [min_calls](#)
Minimum number of calls to estimate the variance (default 100).
- size_t [min_calls_per_bisection](#)
Minimum number of calls required to proceed with bisection (default 4000).

Protected Member Functions

- virtual int [estimate_corrmc](#) (func_t &func, size_t ndim, const vec_t &xl, const vec_t &xu, param_t &pa, size_t calls, double &res, double &err, const double lxmid[], double lsigma_l[], double lsigma_r[])
Desc.

Protected Attributes

- size_t [dim](#)
Desc.
- double * [xmid](#)
Desc.
- double * [sigma_l](#)
Desc.
- double * [sigma_r](#)
Desc.
- double * [fmax_l](#)
Desc.
- double * [fmax_r](#)
Desc.
- double * [fmin_l](#)
Desc.
- double * [fmin_r](#)
Desc.
- double * [fsum_l](#)
Desc.
- double * [fsum_r](#)
Desc.
- double * [fsum2_l](#)
Desc.
- double * [fsum2_r](#)
Desc.
- size_t * [hits_l](#)
Desc.
- size_t * [hits_r](#)
Desc.
- alloc_t [ao](#)
Memory allocator.
- alloc_vec_t [x](#)
The most recent integration point.

8.136.2 Field Documentation

8.136.2.1 double alpha

How estimated variances for two sub-regions are combined.

From GSL documentation:

This parameter controls how the estimated variances for the two sub-regions of a bisection are combined when allocating points. With recursive sampling the overall variance should scale better than $1/N$, since the values from the sub-regions will be obtained using a procedure which explicitly minimizes their variance. To accommodate this behavior the MISER algorithm allows the total variance to depend on a scaling parameter α ,

$$\text{Var}(f) = \{\sigma_a \text{ over } N_a^\alpha\} + \{\sigma_b \text{ over } N_b^\alpha\}.$$

The authors of the original paper describing MISER recommend the value $\alpha = 2$ as a good choice, obtained from numerical experiments, and this is used as the default value in this implementation.

Definition at line 125 of file `gsl_miser.h`.

8.136.2.2 double dither

Introduce random variation into bisection (default 0.0).

From GSL documentation:

This parameter introduces a random fractional variation of size DITHER into each bisection, which can be used to break the symmetry of integrands which are concentrated near the exact center of the hypercubic integration region. The default value of dither is zero, so no variation is introduced. If needed, a typical value of DITHER is 0.1.

Definition at line 90 of file `gsl_miser.h`.

8.136.2.3 double estimate_frac

Specify fraction of function calls for estimating variance.

From GSL documentation:

This parameter specifies the fraction of the currently available number of function calls which are allocated to estimating the variance at each recursive step. The default value is 0.1.

Definition at line 102 of file `gsl_miser.h`.

8.136.2.4 size_t min_calls

Minimum number of calls to estimate the variance (default 100).

From GSL documentation:

This parameter specifies the minimum number of function calls required for each estimate of the variance. If the number of function calls allocated to the estimate using ESTIMATE_FRAC falls below MIN_CALLS then MIN_CALLS are used instead. This ensures that each estimate maintains a reasonable level of accuracy. The default value of MIN_CALLS is `'16 * dim'`.

Definition at line 141 of file `gsl_miser.h`.

8.136.2.5 `size_t min_calls_bisection`

Minimum number of calls required to proceed with bisection (default 4000).

From GSL documentation:

This parameter specifies the minimum number of function calls required to proceed with a bisection step. When a recursive step has fewer calls available than `MIN_CALLS_PER_BISECTION` it performs a plain Monte Carlo estimate of the current sub-region and terminates its branch of the recursion. The default value of this parameter is `'32 * min_calls'`.

Definition at line 157 of file `gsl_miser.h`.

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

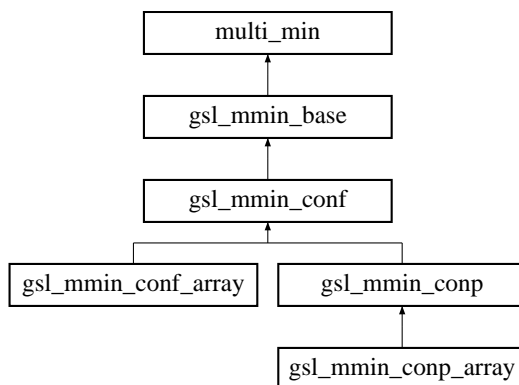
- `gsl_miser.h`

8.137 `gsl_mmin_base` Class Template Reference

Base minimization routines for `gsl_mmin_conf` and `gsl_mmin_conp`.

```
#include <gsl_mmin_conf.h>
```

Inheritance diagram for `gsl_mmin_base`:



8.137.1 Detailed Description

```
template<class param_t, class func_t = multi_func_t<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_func_t<param_t, ovector_base>, class auto_grad_t = gradient<param_t, func_t, ovector_base>, class def_auto_grad_t = simple_grad<param_t, func_t, ovector_base>> 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`.

This class is used by the `gsl_mmin_conf` and `gsl_mmin_conp` minimizers to perform the line minimization along a specified direction. It is not intended for a casual end-user.

Default template arguments

- `param_t` - no default
- `func_t` - `multi_func_t<param_t>`

- `vec_t` - [ovector_base](#)
- `alloc_vec_t` - [ovector](#)
- `alloc_t` - [ovector_alloc](#)
- `dfunc_t` - [grad_funct](#)<param_t,ovector_base>
- `auto_grad_t` - [gradient](#)<param_t,func_t,ovector_base>
- `def_auto_grad_t` - [simple_grad](#)<param_t,func_t,ovector_base>

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

Public Member Functions

- `int base_set` (func_t &ufunc, param_t &pa, auto_grad_t &u_def_grad)
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](#)
Temporary vector.

8.137.2 Member Function Documentation

8.137.2.1 `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 $f_a > f_b < f_c$. Choose an initial xb based on parabolic interpolation

Definition at line 123 of file `gsl_mmin_conf.h`.

8.137.2.2 `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 (x0, f0) 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 .

Idea for future

Remove goto statements

Definition at line 179 of file `gsl_mmin_conf.h`.

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

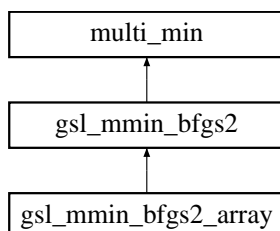
- `gsl_mmin_conf.h`

8.138 gsl_mmin_bfgs2 Class Template Reference

Multidimensional minimization by the BFGS algorithm (GSL).

```
#include <gsl_mmin_bfgs2.h>
```

Inheritance diagram for `gsl_mmin_bfgs2::`



8.138.1 Detailed Description

```
template<class param_t, class func_t = multi_func_t<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc, class dfunc_t = grad_func_t<param_t, ovector_base>, class auto_grad_t = gradient<param_t, func_t,
ovector_base>, class def_auto_grad_t = simple_grad<param_t, func_t, ovector_base>> class gsl_mmin_bfgs2< param_t,
func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >
```

Multidimensional minimization by the BFGS algorithm (GSL).

The functions `mmin()` and `mmin_de()` minimize a given function until the gradient is smaller than the value of `multi_min::tolf` (which defaults to 10^{-4}).

See an example for the usage of this class in [Multidimensional minimizer example](#).

This class includes the optimizations from the GSL minimizer `vector_bfgs2`.

Definition at line 400 of file `gsl_mmin_bfgs2.h`.

Public Member Functions

- 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 `set_de` (vec_t &x, double u_step_size, double tol_u, func_t &ufunc, dfunc_t &udfunc, param_t &upa)
Set the function, the [gradient](#), and the 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 nn.
- 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 nn.

Data Fields

- double `step_size`
The size of the first trial step.
- double `lmin_tol`
The tolerance for the 1-dimensional minimizer.
- def_auto_grad_t `def_grad`
Default automatic [gradient](#) object.

Protected Attributes

- `gsl_mmin_linmin lm`
The line minimizer.
- size_t `dim`
Memory size.
- alloc_t `ao`
Memory allocation.
- auto_grad_t * `agrad`

Automatic [gradient](#) object.

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, auto_grad_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**
- alloc_vec_t **st_grad**
- double **st_f**

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

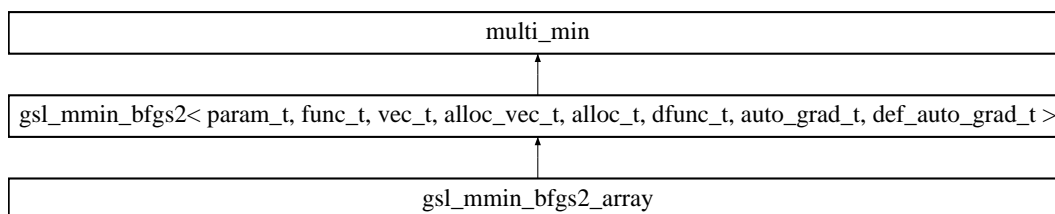
- [gsl_mmin_bfgs2.h](#)

8.139 gsl_mmin_bfgs2_array Class Template Reference

An array version of [gsl_mmin_bfgs2](#).

```
#include <gsl_mmin_bfgs2.h>
```

Inheritance diagram for `gsl_mmin_bfgs2_array`:



8.139.1 Detailed Description

```
template<class param_t, size_t nv> class gsl_mmin_bfgs2_array< param_t, nv >
```

An array version of [gsl_mmin_bfgs2](#).

Definition at line 892 of file `gsl_mmin_bfgs2.h`.

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

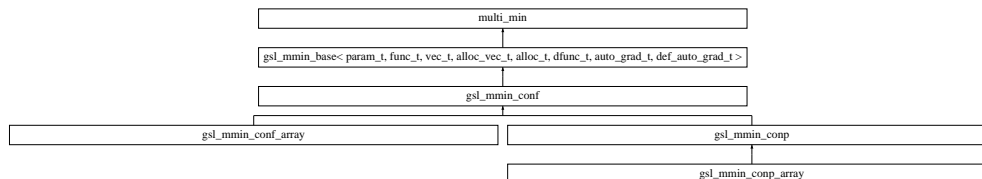
- [gsl_mmin_bfgs2.h](#)

8.140 gsl_mmin_conf Class Template Reference

Multidimensional minimization by the Fletcher-Reeves conjugate gradient algorithm (GSL).

```
#include <gsl_mmin_conf.h>
```

Inheritance diagram for gsl_mmin_conf::



8.140.1 Detailed Description

template<class param_t, class func_t = multi_funct<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<param_t,ovector_base>, class auto_grad_t = gradient<param_t,func_t,ovector_base>, class def_auto_grad_t = simple_grad<param_t,func_t,ovector_base>> 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 functions [mmin\(\)](#) and [mmin_de\(\)](#) minimize a given function until the gradient is smaller than the value of [multi_min::tolf](#) (which defaults to 10^{-4}).

See an example for the usage of this class in [Multidimensional minimizer example](#).

Idea for future

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`;

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.

Default template arguments

- `param_t` - no default
- `func_t` - [multi_funct](#)<param_t>
- `vec_t` - [ovector_base](#)
- `alloc_vec_t` - [ovector](#)
- `alloc_t` - [ovector_alloc](#)
- `dfunc_t` - [grad_funct](#)<param_t,ovector_base>
- `auto_grad_t` - [gradient](#)<param_t,func_t,ovector_base>
- `def_auto_grad_t` - [simple_grad](#)<param_t,func_t,ovector_base>

Definition at line 415 of file `gsl_mmin_conf.h`.

Public Member Functions

- 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 (default 0.01).

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](#)
Iteration number.
- double [step](#)
Stepsize.
- double [tol](#)
Tolerance.
- 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](#)
Proposed minimum.
- gsl_vector * [ugg](#)
Gradient.
- gsl_vector * [udx](#)
Proposed step.
- double [it_min](#)
Desc.

8.140.2 Member Function Documentation

8.140.2.1 virtual int allocate (size_t n) [inline, virtual]

Allocate the memory.

Idea for future

Use a `gsl_alloc_arrays()` like function for this (but keep in mind these are `calloc`, not `malloc` statements)

Definition at line 600 of file `gsl_mmin_conf.h`.

8.140.2.2 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 717 of file `gsl_mmin_conf.h`.

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

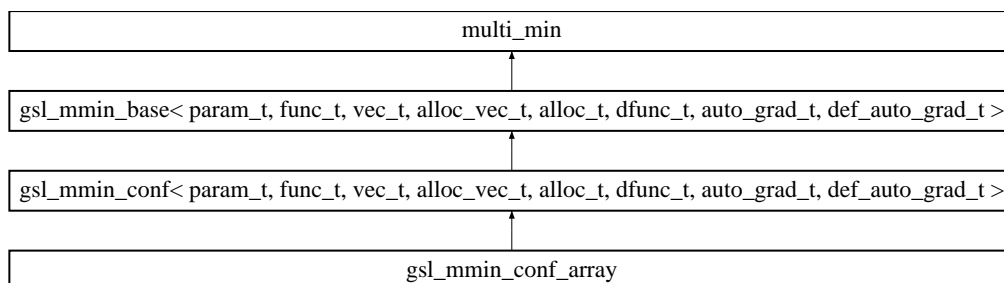
- `gsl_mmin_conf.h`

8.141 gsl_mmin_conf_array Class Template Reference

An array version of [gsl_mmin_conf](#).

```
#include <gsl_mmin_conf.h>
```

Inheritance diagram for `gsl_mmin_conf_array`:



8.141.1 Detailed Description

template<class param_t, size_t nv> class gsl_mmin_conf_array< param_t, nv >

An array version of [gsl_mmin_conf](#).

Default template arguments

- param_t - no default
- func_t - [multi_vfunct](#)<param_t>
- vec_t - double [nv]
- alloc_vec_t - double [nv]
- alloc_t - [array_alloc](#)<double[nv]>
- dfunc_t - [grad_vfunct](#)<param_t,nv>
- auto_grad_t - [gradient_array](#) <param_t,[multi_vfunct](#)<param_t,nv>,nv>
- def_auto_grad_t - [simple_grad](#) <param_t,[multi_vfunct](#)<param_t,nv>,nv>

Definition at line 924 of file [gsl_mmin_conf.h](#).

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

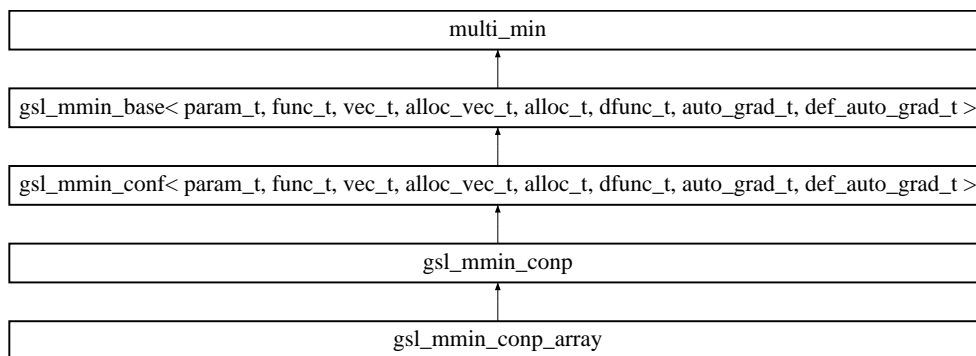
- [gsl_mmin_conf.h](#)

8.142 gsl_mmin_conp Class Template Reference

Multidimensional minimization by the Polak-Ribiere conjugate [gradient](#) algorithm (GSL).

`#include <gsl_mmin_conp.h>`

Inheritance diagram for `gsl_mmin_conp`:



8.142.1 Detailed Description

template<class param_t, class func_t = multi_funct<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<param_t,ovector_base>, class auto_grad_t = gradient<param_t,func_t,ovector_base>, class def_auto_grad_t = simple_grad<param_t,func_t,ovector_base>> class gsl_mmin_conp< param_t, func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

Multidimensional minimization by the Polak-Ribiere conjugate [gradient](#) algorithm (GSL).

The functions [mmin\(\)](#) and [mmin_de\(\)](#) minimize a given function until the gradient is smaller than the value of [multi_min::tolf](#) (which defaults to 10^{-4}).

See an example for the usage of this class in [Multidimensional minimizer example](#).

Idea for future

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.

Definition at line 55 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:

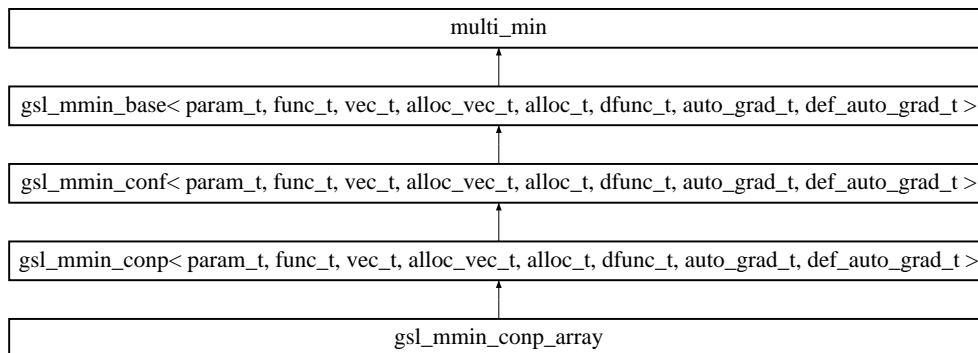
- `gsl_mmin_conp.h`

8.143 gsl_mmin_conp_array Class Template Reference

An array version of [gsl_mmin_conp](#).

```
#include <gsl_mmin_conp.h>
```

Inheritance diagram for `gsl_mmin_conp_array`:



8.143.1 Detailed Description

```
template<class param_t, size_t nv> class gsl_mmin_conp_array< param_t, nv >
```

An array version of [gsl_mmin_conp](#).

Definition at line 192 of file `gsl_mmin_conp.h`.

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

- `gsl_mmin_conp.h`

8.144 gsl_mmin_linmin Class Reference

The line minimizer for [gsl_mmin_bfgs2](#).

```
#include <gsl_mmin_bfgs2.h>
```

8.144.1 Detailed Description

The line minimizer for [gsl_mmin_bfgs2](#).

Definition at line 317 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 z1, 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 z1, 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.

8.144.2 Member Function Documentation

8.144.2.1 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 $\eta=3*(f_1-f_0)-2*fp_0-fp_1$, $\xi=fp_0+fp_1-2*(f_1-f_0)$.

8.144.2.2 double interp_quad (double f0, double fp0, double f1, double z1, 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 $q(x) = f_0 + fp_0 * x + (f_1-f_0-fp_0) * x^2$

8.144.2.3 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 [Fletcher87](#) 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:

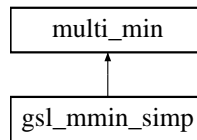
- `gsl_mmin_bfgs2.h`

8.145 gsl_mmin_simp Class Template Reference

Multidimensional minimization by the Simplex method (GSL).

```
#include <gsl_mmin_simp.h>
```

Inheritance diagram for `gsl_mmin_simp`:



8.145.1 Detailed Description

```
template<class param_t, class func_t = multi_func_t<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_mmin_simp< param_t, func_t, vec_t, alloc_vec_t, alloc_t >
```

Multidimensional minimization by the Simplex method (GSL).

This class minimizes a function using Nelder and Mead's Simplex algorithm. A simplex in a N-dimensional space is defined as a set of N+1 points which describe an N-dimensional volume surrounding the minimum. The algorithm proceeds by shifting the simplex points until the simplex is sufficiently small and thus the minimum is known with sufficient accuracy.

For a slightly improved method, see [gsl_mmin_simp2](#).

This class has a high-level interface using [mmin\(\)](#), [mmin_twovec\(\)](#) or [mmin_simplex\(\)](#) which automatically performs the memory allocation and minimization, or a GSL-like interface using [allocate\(\)](#), [free\(\)](#), [iterate\(\)](#) and [set\(\)](#) or [set_simplex\(\)](#).

The simplex can be completely specified by the user (see [mmin_simplex\(\)](#) and [set_simplex\(\)](#)). Alternatively, the simplex is automatically specified given initial guess x_j and a step size vector s_k for $0 \leq k < n_s$. The simplex p_{ij} with $0 \leq i \leq n$ and $0 \leq j < n$ is chosen with $p_{0j} = x_j$ and

$$\begin{aligned}
 p_{i+1,j} &= x_j \quad \text{for } i \neq j \\
 p_{i+1,j} &= x_j + s_{j \bmod n_s} \quad \text{for } i = j
 \end{aligned}$$

for $0 < i < n$. The step size vector s is set by the [set_step\(\)](#) member function. The presence of mod in the recipe above just indicates that elements of the step size vector are automatically re-used if there are less step sizes than dimensions in the minimization problem.

Based on [Nelder65](#).

Definition at line 93 of file `gsl_mmin_simp.h`.

Public Member Functions

- `template<class vec2_t >`
`int set_step (size_t nv, vec2_t &step)`
Set the step sizes for each independent variable.
- `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.

- template<class mat_t >
int [mmin_simplex](#) (size_t nn, mat_t &sx, double &fmin, param_t &pa, func_t &ufunc)
Calculate the minimum min of func w.r.t the array x of size nvar, given an initial 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, size_t n, vec_t &ax, vec_t &step_size)
Set the function and initial guess.
- template<class mat_t >
int [set_simplex](#) (func_t &ufunc, param_t &pa, mat_t &sx)
Set the function and initial simplex.
- virtual int [iterate](#) ()
Perform an iteration.
- virtual int [print_iter](#) (size_t nv, vec_t &xx, alloc_vec_t *simp, 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 [size](#)
Size of current simplex computed by [iterate\(\)](#).
- alloc_vec_t [x](#)
Present minimum vector computed by [iterate\(\)](#).
- double [fval](#)
Function value at minimum computed by [iterate\(\)](#).
- int [print_simplex](#)
Print simplex information in [print_iter\(\)](#) (default 0).

Protected Member Functions

- int [nmsimplex_calc_center](#) (vec_t &mp)
Compute the center of the simplex and store in mp.
- double [nmsimplex_size](#) ()
Compute the size of the simplex.
- virtual int [move_corner_err](#) (const double coeff, size_t corner, vec_t &xc, func_t &f, size_t nvar, param_t &pa, double &newval)
Move a corner of a simplex.
- virtual int [contract_by_best](#) (size_t best, vec_t &xc, func_t &f, size_t nvar, param_t &pa)
Contract the simplex towards the best point.

Protected Attributes

- alloc_vec_t * [x1](#)
An array of n+1 vectors containing the simplex.
- [ovector](#) [y1](#)
The n+1 function values at the simplex points.
- alloc_vec_t [ws1](#)
Workspace vector 1.
- alloc_vec_t [ws2](#)
Workspace vector 2.
- alloc_vec_t [ws3](#)
Workspace vector 3.

- size_t **dim**
Number of variables to be minimized over.
- func_t * **func**
Function.
- param_t * **params**
Parameters.
- bool **set_called**
True if `set()` has been called.
- ovector **step_vec**
Vector of step sizes.
- alloc_t **ao**
Vector allocator.
- bool **avoid_nonzero**
If true, try to automatically avoid regions where the function returns a non-zero value (default false).

8.145.2 Member Function Documentation

8.145.2.1 virtual int contract_by_best (size_t best, vec_t & 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 vector, xc, is used as work space.

Definition at line 201 of file gsl_mmin_simp.h.

8.145.2.2 virtual int move_corner_err (const double coeff, size_t corner, vec_t & xc, func_t & f, size_t nvar, param_t & pa, double & newval) [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 in newval.

Definition at line 172 of file gsl_mmin_simp.h.

8.145.2.3 double nmsimplex_size () [inline, protected]

Compute the size of the simplex.

Calculates simplex size as average sum of length of vectors from simplex center to corner points:

$$(1/n) \sum ||y - y_{\text{middlepoint}}||$$

Definition at line 150 of file gsl_mmin_simp.h.

8.145.2.4 virtual int print_iter (size_t nv, vec_t & xx, alloc_vec_t * simp, 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 756 of file gsl_mmin_simp.h.

8.145.3 Field Documentation

8.145.3.1 bool avoid_nonzero [protected]

If true, try to automatically avoid regions where the function returns a non-zero value (default false).

Note:

This option doesn't work yet, so I've made the variable protected to prevent the user from changing it.

Definition at line 279 of file `gsl_mmin_simp.h`.

8.145.3.2 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. If this is 2 and `verbose` is greater than 0, then `print_iter()` will print the simplex coordinates in addition to the function values.

Definition at line 326 of file `gsl_mmin_simp.h`.

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

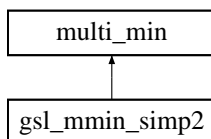
- `gsl_mmin_simp.h`

8.146 gsl_mmin_simp2 Class Template Reference

Multidimensional minimization by the Simplex method (v2) (GSL).

```
#include <gsl_mmin_simp2.h>
```

Inheritance diagram for `gsl_mmin_simp2::`



8.146.1 Detailed Description

```
template<class param_t, class func_t = multi_func<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_mmin_simp2< param_t, func_t, vec_t, alloc_vec_t, alloc_t >
```

Multidimensional minimization by the Simplex method (v2) (GSL).

Bug

This class doesn't work at the moment, and it's not yet clear if this bug is limited to the O₂scl version or may also be present in GSL.

This class minimizes a function using Nelder and Mead's Simplex algorithm. A simplex in a N-dimensional space is defined as a set of N+1 points which describe an N-dimensional volume surrounding the minimum. The algorithm proceeds by shifting the simplex points until the simplex is sufficiently small and thus the minimum is known with sufficient accuracy.

For the earlier method used in GSL, see [gsl_mmin_simp](#).

This class has a high-level interface using [mmin\(\)](#), [mmin_twovec\(\)](#) or [mmin_simplex\(\)](#) which automatically performs the memory allocation and minimization, or a GSL-like interface using [allocate\(\)](#), [free\(\)](#), [iterate\(\)](#) and [set\(\)](#) or [set_simplex\(\)](#).

The simplex can be completely specified by the user (see [mmin_simplex\(\)](#) and [set_simplex\(\)](#)). Alternatively, the simplex is automatically specified given initial guess x_j and a step size vector s_k for $0 \leq k < n_s$. The simplex p_{ij} with $0 \leq i \leq n$ and $0 \leq j < n$ is chosen with $p_{0j} = x_j$ and

$$\begin{aligned} p_{i+1,j} &= x_j \quad \text{for } i \neq j \\ p_{i+1,j} &= x_j + s_{j \bmod n_s} \quad \text{for } i = j \end{aligned}$$

for $0 < i < n$. The step size vector s is set by the [set_step\(\)](#) member function. The presence of mod in the recipe above just indicates that elements of the step size vector are automatically re-used if there are less step sizes than dimensions in the minimization problem.

See an example for the usage of this class in [Multidimensional minimizer example](#).

Based on [Nelder65](#).

Definition at line 100 of file `gsl_mmin_simp2.h`.

Public Member Functions

- `template<class vec2_t >`
`int set_step (size_t nv, vec2_t &step)`
Set the step sizes for each independent variable.
- `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.
- `template<class mat_t >`
`int mmin_simplex (size_t nn, mat_t &sx, double &fmin, param_t &pa, func_t &ufunc)`
Calculate the minimum min of func w.r.t the array x of size nvar, given an initial 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, size_t n, vec_t &ax, vec_t &step_size)`
Set the function and initial guess.
- `template<class mat_t >`
`int set_simplex (func_t &ufunc, param_t &pa, mat_t &sx)`
Set the function and initial simplex.
- `virtual int iterate ()`
Perform an iteration.
- `virtual int print_iter (size_t nv, vec_t &xx, alloc_vec_t *simp, 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_simp2").

Data Fields

- `double size`
Size of current simplex computed by [iterate\(\)](#).
- `alloc_vec_t x`
Present minimum vector computed by [iterate\(\)](#).
- `double fval`
Function value at minimum computed by [iterate\(\)](#).
- `int print_simplex`
Print simplex information in [print_iter\(\)](#) (default 0).

Protected Member Functions

- int `compute_center` ()
Compute the center of the simplex.
- double `compute_size` ()
Compute the size of the simplex.
- virtual int `try_corner_move` (const double coeff, size_t corner, vec_t &xc, func_t &f, size_t nvar, param_t &pa, double &new-val)
Move a corner of a simplex.
- virtual int `update_point` (size_t i, vec_t &xx, double val)
Desc.
- virtual int `contract_by_best` (size_t best, func_t &f, size_t nvar, param_t &pa)
Contract the simplex towards the best point.

Protected Attributes

- alloc_vec_t * `x1`
An array of $n+1$ vectors containing the simplex.
- ovector `y1`
The $n+1$ function values at the simplex points.
- alloc_vec_t `ws1`
Workspace vector 1.
- alloc_vec_t `ws2`
Workspace vector 2.
- alloc_vec_t `ws3`
Workspace vector 3.
- alloc_vec_t `center`
Center of simplex.
- alloc_vec_t `delta`
Desc.
- alloc_vec_t `xmc`
Distance of vector from center.
- double `S2`
Squared simplex size.
- size_t `dim`
Number of variables to be minimized over.
- func_t * `func`
Function.
- param_t * `params`
Parameters.
- bool `set_called`
True if `set()` has been called.
- ovector `step_vec`
Vector of step sizes.
- alloc_t `ao`
Vector allocator.
- bool `avoid_nonzero`
If true, try to automatically avoid regions where the function returns a non-zero value (default false).

8.146.2 Member Function Documentation

8.146.2.1 double compute_size () [inline, protected]

Compute the size of the simplex.

Calculates simplex size as average sum of length of vectors from simplex center to corner points:

$$(1/n) \sum ||y - y_{\text{middlepoint}}||$$

Definition at line 165 of file `gsl_mmin_simp2.h`.

8.146.2.2 virtual int contract_by_best (size_t *best*, 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.

Definition at line 261 of file `gsl_mmin_simp2.h`.

8.146.2.3 virtual int print_iter (size_t *nv*, vec_t & *xx*, alloc_vec_t * *simp*, 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 823 of file `gsl_mmin_simp2.h`.

8.146.2.4 virtual int try_corner_move (const double *coeff*, size_t *corner*, vec_t & *xc*, func_t & *f*, size_t *nvar*, param_t & *pa*, double & *newval*) [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` in `newval`.

Definition at line 190 of file `gsl_mmin_simp2.h`.

8.146.3 Field Documentation

8.146.3.1 bool avoid_nonzero [protected]

If true, try to automatically avoid regions where the function returns a non-zero value (default false).

Note:

This option doesn't work yet, so I've made the variable protected to prevent the user from changing it.

Definition at line 346 of file `gsl_mmin_simp2.h`.

8.146.3.2 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. If this is 2 and `verbose` is greater than 0, then `print_iter()` will print the simplex coordinates in addition to the function values.

Definition at line 393 of file `gsl_mmin_simp2.h`.

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

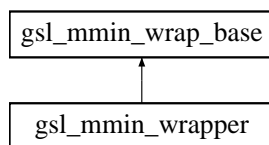
- `gsl_mmin_simp2.h`

8.147 gsl_mmin_wrap_base Class Reference

Virtual base for the `gsl_mmin_bfgs2` wrapper.

```
#include <gsl_mmin_bfgs2.h>
```

Inheritance diagram for `gsl_mmin_wrap_base`:



8.147.1 Detailed Description

Virtual base for the `gsl_mmin_bfgs2` wrapper.

This class 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 44 of file `gsl_mmin_bfgs2.h`.

Public Member Functions

- virtual double `wrap_f` (double alpha, int params)=0
Function.
- virtual double `wrap_df` (double alpha, int params)=0
Derivative.
- virtual void `wrap_fdf` (double alpha, int params, double *f, double *df)=0
Function and derivative.

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

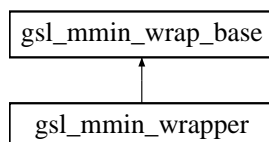
- `gsl_mmin_bfgs2.h`

8.148 `gsl_mmin_wrapper` Class Template Reference

Wrapper class for the `gsl_mmin_bfgs2` minimizer.

```
#include <gsl_mmin_bfgs2.h>
```

Inheritance diagram for `gsl_mmin_wrapper`:



8.148.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc,
class dfunc_t = grad_func_t<param_t,ovector_base>, class auto_grad_t = gradient<param_t,func_t,ovector_base>> class
gsl_mmin_wrapper< param_t, func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t >
```

Wrapper class for the `gsl_mmin_bfgs2` minimizer.

This is a reimplement of the internal GSL wrapper for function calls in the BFGS minimizer.

Idea for future

There's a bit of extra vector copying here which could potentially be avoided.

Definition at line 68 of file gsl_mmin_bfgs2.h.

Public Member Functions

- void [prepare_wrapper](#) (func_t &ufunc, dfunc_t *udfunc, param_t &upa, gsl_vector *t_x, double f, gsl_vector *t_g, gsl_vector *t_p, auto_grad_t *ag)
Initialize wrapper.
- void [update_position](#) (double alpha, vec_t &t_x, double *t_f, vec_t &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, int params)
Evaluate the function.
- virtual double [wrap_df](#) (double alpha, int params)
Evaluate the derivative.
- virtual void [wrap_fdf](#) (double alpha, int params, double *f, double *df)
Evaluate the function and the derivative.

Protected Attributes

- func_t * [func](#)
Function.
- dfunc_t * [dfunc](#)
Derivative.
- auto_grad_t * [agrad](#)
The automatic [gradient](#) object.
- param_t * [pa](#)
Parameters.
- bool [grad_given](#)
True if the [gradient](#) was given by the user.

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`

8.148.2 Member Function Documentation

8.148.2.1 `void change_direction() [inline]`

Convert cache values to the new minimizer direction.

Convert the cache values from the end of the current minimisation to those needed for the start of the next minimisation, $\alpha=0$

Definition at line 289 of file `gsl_mmin_bfgs2.h`.

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

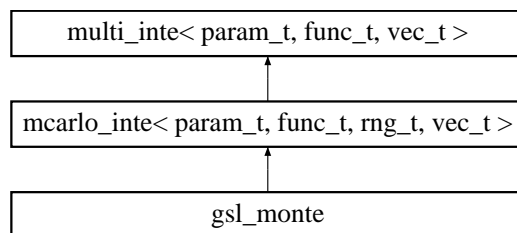
- `gsl_mmin_bfgs2.h`

8.149 gsl_monte Class Template Reference

Multidimensional integration using plain Monte Carlo (GSL).

```
#include <gsl_monte.h>
```

Inheritance diagram for `gsl_monte`:



8.149.1 Detailed Description

```
template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_monte< param_t, func_t, rng_t, vec_t, alloc_vec_t, alloc_t >
```

Multidimensional integration using plain Monte Carlo (GSL).

Definition at line 62 of file `gsl_monte.h`.

Public Member Functions

- 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 double [minteg](#) (func_t &func, size_t ndim, const vec_t &a, const vec_t &b, param_t &pa)
Integrate function func over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i < ndim-1$.
- virtual const char * [type](#) ()
Return string denoting type ("gsl_monte").

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

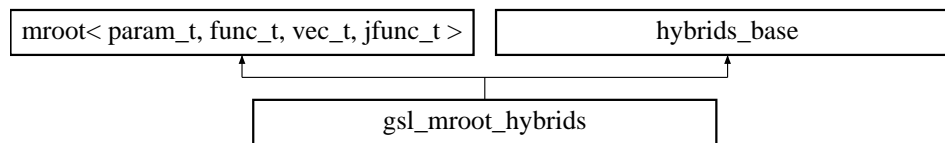
- [gsl_monte.h](#)

8.150 gsl_mroot_hybrids Class Template Reference

Multidimensional root-finding algorithm using Powell's Hybrid method (GSL).

```
#include <gsl_mroot_hybrids.h>
```

Inheritance diagram for `gsl_mroot_hybrids`:



8.150.1 Detailed Description

`template<class param_t, class func_t = mm_func_t<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc, class jfunc_t = jac_func_t<param_t,vec_t,mat_t>> class gsl_mroot_hybrids< param_t, func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_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 and the memory allocation 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 $\sum_i |f_i| < \text{mroot::tol}$ 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.

The default method for numerically computing the Jacobian is from [simple_jacobian](#). This default is identical to the GSL approach, except that the default value of [simple_jacobian::epsmin](#) is non-zero. See [simple_jacobian](#) for more details.

There is an example for the usage of the multidimensional solver classes given in `examples/ex_mroot.cpp`, see the [Multidimensional solver example](#).

Idea for future

It's kind of strange that [set\(\)](#) sets `jac_given` to false and [set_de\(\)](#) has to reset it to true. Can this be simplified?

Definition at line 352 of file gsl_mroot_hybrids.h.

Public Member Functions

- virtual int [set_jacobian](#) ([jacobian](#)< param_t, func_t, vec_t, mat_t > &j)
Set the automatic Jacobian object.
- int [iterate](#) ()
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, mat_t, alloc_vec_t, alloc_t > [def_jac](#)
Default automatic Jacobian object.
- alloc_vec_t [f](#)
The value of the function at the present iteration.
- alloc_vec_t [x](#)
The present solution.

Protected Member Functions

- void [compute_Rg](#) (size_t N, const gsl_matrix *r, const gsl_vector *gradient, vec_t &Rg)
Desc.
- void [compute_wv](#) (size_t n, const gsl_vector *qtdf, const gsl_vector *rdx, const vec_t &dxx, const gsl_vector *diag, double pnorm, gsl_vector *w, gsl_vector *v)
Desc.
- void [compute_rdx](#) (size_t N, const gsl_matrix *r, const vec_t &dxx, gsl_vector *rdx)
Desc.
- double [scaled_enorm_tvec](#) (size_t n, const gsl_vector *d, const vec_t &ff)
Desc.
- double [compute_delta](#) (size_t n, gsl_vector *diag, vec_t &xx)
Desc.
- double [enorm_tvec](#) (const vec_t &ff)
Desc.
- int [compute_trial_step_tvec](#) (size_t N, vec_t &xl, vec_t &dxl, vec_t &xx_trial)
Desc.
- int [compute_df_tvec](#) (size_t n, const vec_t &ff_trial, const vec_t &fl, gsl_vector *dffl)
Desc.
- void [compute_diag_tvec](#) (size_t n, const mat_t &J, gsl_vector *diag)

Desc.

- void [compute_qtf_tvec](#) (size_t N, const gsl_matrix *q, const vec_t &ff, gsl_vector *qtf)

Desc.

- void [update_diag_tvec](#) (size_t n, const mat_t &J, gsl_vector *diag)

Desc.

- void [scaled_addition_tvec](#) (size_t N, double alpha, gsl_vector *newton, double beta, gsl_vector *[gradient](#), vec_t &pp)

Desc.

- int [dogleg](#) (size_t n, const gsl_matrix *r, const gsl_vector *qtf, const gsl_vector *diag, double delta, gsl_vector *newton, gsl_vector *[gradient](#), vec_t &p)

Take a dogleg step.

- 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](#)

The user-specified Jacobian.

- [jacobian](#)< param_t, func_t, vec_t, mat_t > * [ajac](#)

The automatic Jacobian.

- alloc_t [ao](#)

Memory allocator for objects of type alloc_vec_t.

- alloc_vec_t [dx](#)

The value of the derivative.

- alloc_vec_t [x_trial](#)

Trial [root](#).

- alloc_vec_t [f_trial](#)

Trial function value.

- o2scl_hybrid_state_t< vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_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](#)

The user-specified function.

- bool [set_called](#)

True if "[set](#)" has been called.

8.150.2 Member Function Documentation

8.150.2.1 int iterate () [inline]

Perform an iteration.

At the end of the iteration, the current value of the solution is stored in [x](#).

Definition at line 746 of file [gsl_mroot_hybrids.h](#).

8.150.2.2 virtual int msolve_de (size_t nn, vec_t & xx, param_t & pa, func_t & ufunc, jfunc_t & dfunc) [inline, virtual]

Solve [func](#) with derivatives [dfunc](#) using [x](#) as an initial guess, returning [x](#).

Reimplemented from [mroot](#).

Definition at line 1014 of file [gsl_mroot_hybrids.h](#).

8.150.3 Field Documentation

8.150.3.1 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 711 of file `gsl_mroot_hybrids.h`.

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

- `gsl_mroot_hybrids.h`

8.151 gsl_ode_control Class Template Reference

Control structure for `gsl_astep`.

```
#include <gsl_astep.h>
```

8.151.1 Detailed Description

```
template<class vec_t> class gsl_ode_control< vec_t >
```

Control structure for `gsl_astep`.

This class implements both the "standard" and "scaled" step control methods from GSL. The standard control method is the default. To use the scaled controle, set `standard` to `false` and set the scale for each component using `set_scale()`.

The control object is a four parameter heuristic based on absolute and relative errors `eps_abs` and `eps_rel`, and scaling factors `a_y` and `a_dydt` for the system state $y(t)$ and derivatives $y'(t)$ respectively.

The step-size adjustment procedure for this method begins by computing the desired error level D_i for each component. In the unscaled version,

$$D_i = \text{eps_abs} + \text{eps_rel} \times (\text{a_y}|y_i| + \text{a_dydt} \, h|y'_i|)$$

while in the scaled version the user specifies the scale for each component, s_i ,

$$D_i = \text{eps_abs} \, s_i + \text{eps_rel} \times (\text{a_y}|y_i| + \text{a_dydt} \, h|y'_i|)$$

The desired error level D_i is compared to then observed error $E_i = |\text{yerr}_i|$. If the observed error E exceeds the desired error level D by more than 10 percent for any component then the method reduces the step-size by an appropriate factor,

$$h_{\text{new}} = S \, h_{\text{old}} \left(\frac{E}{D} \right)^{-1/q}$$

where q is the consistency order of the method (e.g. $q = 4$ for 4(5) embedded RK), and S is a safety factor of 0.9. The ratio E/D is taken to be the maximum of the ratios E_i/D_i .

If the observed error E is less than 50 percent of the desired error level D for the maximum ratio E_i/D_i then the algorithm takes the opportunity to increase the step-size to bring the error in line with the desired level,

$$h_{\text{new}} = S \, h_{\text{old}} \left(\frac{E}{D} \right)^{-1/(q+1)}$$

This encompasses all the standard error scaling methods. To avoid uncontrolled changes in the stepsize, the overall scaling factor is limited to the range 1/5 to 5.

Definition at line 110 of file `gsl_astep.h`.

Public Member Functions

- template<class svec_t >
int [set_scale](#) (size_t nscal, const svec_t &scale)
Set the scaling for each differential equation.
- virtual int [hadjust](#) (size_t dim, unsigned int ord, const vec_t &y, vec_t &yerr, vec_t &yp, double *h)
Automatically adjust step-size.

Data Fields

- double [eps_abs](#)
Absolute precision (default 10^{-6}).
- double [eps_rel](#)
Relative precision (default 0).
- double [a_y](#)
Function scaling factor (default 1).
- double [a_dydt](#)
Derivative scaling factor (default 0).
- bool [standard](#)
Use standard or scaled algorithm (default true).

Protected Attributes

- size_t [sdim](#)
Number of scalings.
- double * [scale_abs](#)
Scalings.

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

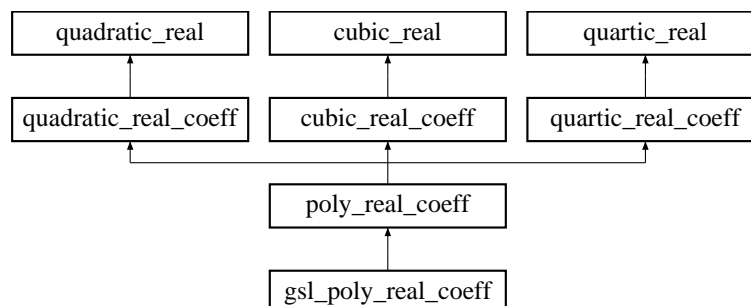
- [gsl_astep.h](#)

8.152 gsl_poly_real_coeff Class Reference

Solve a general polynomial with real coefficients (GSL).

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

Inheritance diagram for `gsl_poly_real_coeff`:



8.152.1 Detailed Description

Solve a general polynomial with real coefficients (GSL).

Definition at line 597 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 $x = x_1, x = x_2, x = x_3$, and $x = x_4$.
- const char * [type](#) ()

Return a string denoting the type ("gsl_poly_real_coeff").

Protected Attributes

- gsl_poly_complex_workspace * [w2](#)

Workspace for quadratic polynomials.
- gsl_poly_complex_workspace * [w3](#)

Workspace for cubic polynomials.
- gsl_poly_complex_workspace * [w4](#)

Workspace for quartic polynomials.
- gsl_poly_complex_workspace * [wgen](#)

Workspace for general polynomials.
- int [gen_size](#)

The size of the workspace [wgen](#).

8.152.2 Member Function Documentation

8.152.2.1 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].

Implements [poly_real_coeff](#).

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

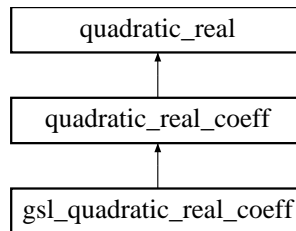
- [poly.h](#)

8.153 gsl_quadratic_real_coeff Class Reference

Solve a quadratic with real coefficients and complex roots (GSL).

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

Inheritance diagram for gsl_quadratic_real_coeff::



8.153.1 Detailed Description

Solve a quadratic with real coefficients and complex roots (GSL).

Definition at line 477 of file `poly.h`.

Public Member Functions

- 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:

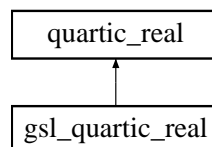
- `poly.h`

8.154 `gsl_quartic_real` Class Reference

Solve a quartic with real coefficients and real roots (GSL).

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

Inheritance diagram for `gsl_quartic_real`:



8.154.1 Detailed Description

Solve a quartic with real coefficients and real roots (GSL).

This class internally uses the GSL functions to solve the resolvent cubic and associated quadratics, while `gsl_quartic_real2` contains explicit code to solve them instead.

Idea for future

Optimize value of `cube_root_tol` and compare more clearly to `gsl_quartic_real2`

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

Public Member Functions

- 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").

Data Fields

- double [cube_root_tol](#)
A tolerance for determining the proper cube [root](#) (default 10^{-4}).

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

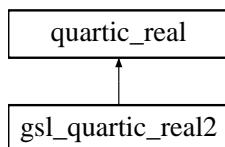
- [poly.h](#)

8.155 gsl_quartic_real2 Class Reference

Solve a quartic with real coefficients and real roots (GSL).

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

Inheritance diagram for gsl_quartic_real2::



8.155.1 Detailed Description

Solve a quartic with real coefficients and real roots (GSL).

This class directly solves resolvent cubic and associated quadratics without using the GSL functions (as done in [gsl_quartic_real](#)).

Idea for future

Optimize value of `cube_root_tol` and compare more clearly to [gsl_quartic_real](#)

Definition at line 569 of file `poly.h`.

Public Member Functions

- 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_real2").

Data Fields

- double [cube_root_tol](#)
A tolerance for determining the proper cube [root](#) (default 10^{-7}).

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

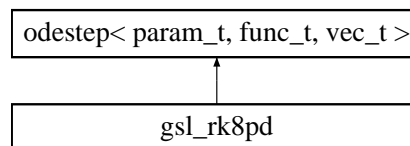
- [poly.h](#)

8.156 gsl_rk8pd Class Template Reference

Embedded Runge-Kutta Prince-Dormand ODE stepper (GSL).

```
#include <gsl_rk8pd.h>
```

Inheritance diagram for gsl_rk8pd::



8.156.1 Detailed Description

template<class param_t, class func_t = ode_func<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_rk8pd< param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Embedded Runge-Kutta Prince-Dormand ODE stepper (GSL).

Based on [Prince81](#) .

Definition at line 59 of file `gsl_rk8pd.h`.

Public Member Functions

- 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]

8.156.2 Member Function Documentation

8.156.2.1 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.

If *derivs* always returns zero, then this function will also return zero. If not, *step()* will return the first non-zero value which was obtained in a call to *derivs*. The error handler is never called.

Implements [odestep](#).

Definition at line 254 of file *gsl_rk8pd.h*.

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

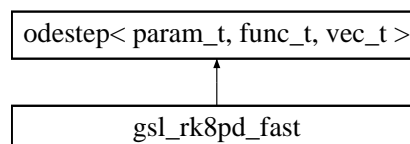
- *gsl_rk8pd.h*

8.157 gsl_rk8pd_fast Class Template Reference

Faster embedded Runge-Kutta Prince-Dormand ODE stepper (GSL).

```
#include <gsl_rk8pd.h>
```

Inheritance diagram for *gsl_rk8pd_fast*:



8.157.1 Detailed Description

template<size_t N, class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_rk8pd_fast< N, param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Faster embedded Runge-Kutta Prince-Dormand ODE stepper (GSL).

This is 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()` should 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.

Based on [Prince81](#) .

Definition at line 426 of file `gsl_rk8pd.h`.

Public Member Functions

- 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]

8.157.2 Member Function Documentation

8.157.2.1 `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` should be equal to the template parameter `N`.

Implements [odestep](#).

Definition at line 628 of file `gsl_rk8pd.h`.

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

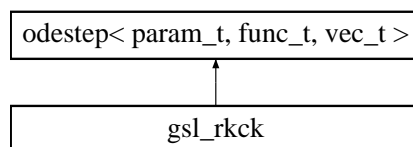
- `gsl_rk8pd.h`

8.158 gsl_rkck Class Template Reference

Cash-Karp embedded Runge-Kutta ODE stepper (GSL).

```
#include <gsl_rkck.h>
```

Inheritance diagram for `gsl_rkck`:



8.158.1 Detailed Description

`template<class param_t, class func_t = ode_func_t<param_t>, class vec_t = ovector_base, 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 ODE stepper (GSL).

Based on [Cash90](#).

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

Public Member Functions

- `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`

8.158.2 Member Function Documentation

8.158.2.1 `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.

If `derivs` always returns zero, then this function will also return zero. If not, `step()` will return the first non-zero value which was obtained in a call to `derivs`. The error handler is never called.

Implements [odestep](#).

Definition at line 157 of file `gsl_rkck.h`.

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

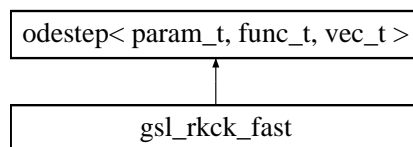
- `gsl_rkck.h`

8.159 gsl_rkck_fast Class Template Reference

Faster Cash-Karp embedded Runge-Kutta ODE stepper (GSL).

```
#include <gsl_rkck.h>
```

Inheritance diagram for `gsl_rkck_fast`:



8.159.1 Detailed Description

template<size_t N, class param_t, class func_t = ode_func<param_t>, class vec_t = ovector_base, 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 >

Faster Cash-Karp embedded Runge-Kutta ODE stepper (GSL).

This is a faster 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()` should 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.

Based on [Cash90](#).

Definition at line 250 of file `gsl_rkck.h`.

Public Member Functions

- 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`

8.159.2 Member Function Documentation

8.159.2.1 `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` should be equal to the template parameter `N`.

Implements [odestep](#).

Definition at line 342 of file `gsl_rkck.h`.

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

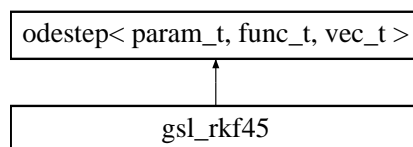
- `gsl_rkck.h`

8.160 gsl_rkf45 Class Template Reference

Runge-Kutta-Fehlberg embedded Runge-Kutta ODE stepper (GSL).

```
#include <gsl_rkf45.h>
```

Inheritance diagram for `gsl_rkf45`:



8.160.1 Detailed Description

`template<class param_t, class func_t = ode_func<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_rkf45< param_t, func_t, vec_t, alloc_vec_t, alloc_t >`

Runge-Kutta-Fehlberg embedded Runge-Kutta ODE stepper (GSL).

Based on [Hairer00](#).

Definition at line 59 of file `gsl_rkf45.h`.

Public Member Functions

- `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 c1`
- `double c3`
- `double c4`
- `double c5`
- `double c6`
- `double ec` [7]

8.160.2 Member Function Documentation

8.160.2.1 `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.

If `derivs` always returns zero, then this function will also return zero. If not, `step()` will return the first non-zero value which was obtained in a call to `derivs`. The error handler is never called.

Implements [odestep](#).

Definition at line 157 of file `gsl_rkf45.h`.

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

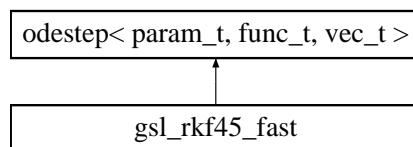
- `gsl_rkf45.h`

8.161 gsl_rkf45_fast Class Template Reference

Faster Runge-Kutta-Fehlberg embedded Runge-Kutta ODE stepper (GSL).

```
#include <gsl_rkf45.h>
```

Inheritance diagram for `gsl_rkf45_fast`:



8.161.1 Detailed Description

template<size_t N, class param_t, class func_t = ode_func<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_rkf45_fast< N, param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Faster Runge-Kutta-Fehlberg embedded Runge-Kutta ODE stepper (GSL).

This is a faster version of [gsl_rkf45](#), 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()` should 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.

Based on [Hairer00](#).

Definition at line 257 of file `gsl_rkf45.h`.

Public Member Functions

- 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 **c1**
- double **c3**
- double **c4**
- double **c5**
- double **c6**
- double **ec** [7]

8.161.2 Member Function Documentation

8.161.2.1 `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` should be equal to the template parameter `N`.

Implements [odestep](#).

Definition at line 355 of file `gsl_rkf45.h`.

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

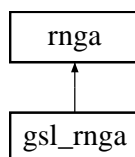
- `gsl_rkf45.h`

8.162 gsl_rnga Class Reference

Random number generator (GSL).

```
#include <gsl_rnga.h>
```

Inheritance diagram for `gsl_rnga`:



8.162.1 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 42 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 generator 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 $[0, \text{max} - 1]$.
- void [set_seed](#) (unsigned long int s)
Set the seed.
- void [clock_seed](#) ()
Set the seed.

Protected Attributes

- gsl_rng * [gr](#)
The GSL random number generator.
- const gsl_rng_type * [rng](#)
The GSL random number generator type.

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

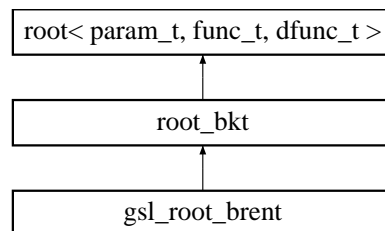
- [gsl_rnga.h](#)

8.163 gsl_root_brent Class Template Reference

One-dimensional root-finding (GSL).

```
#include <gsl_root_brent.h>
```

Inheritance diagram for `gsl_root_brent`:



8.163.1 Detailed Description

```
template<class param_t, class func_t = funct<param_t>> class gsl_root_brent< param_t, func_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.

An example demonstrating the usage of this class is given in `examples/ex_fptr.cpp` and the [Function and solver example](#).

Idea for future

There is some duplication in the variables `x_lower`, `x_upper`, `a`, and `b`, which could be removed.

Definition at line 85 of file `gsl_root_brent.h`.

Public Member Functions

- 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**

8.163.2 Member Function Documentation

8.163.2.1 int iterate (func_t &f) [inline]

Perform an iteration.

This function always returns [gsl_success](#).

Definition at line 108 of file `gsl_root_brent.h`.

8.163.2.2 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 340 of file `gsl_root_brent.h`.

8.163.2.3 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

Implements [root_bkt](#).

Definition at line 220 of file `gsl_root_brent.h`.

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

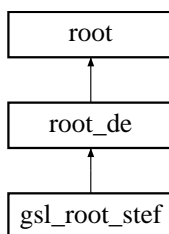
- `gsl_root_brent.h`

8.164 gsl_root_stef Class Template Reference

Steffenson equation solver (GSL).

```
#include <gsl_root_stef.h>
```

Inheritance diagram for `gsl_root_stef`:

**8.164.1 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 is Newton's method with an Aitken "delta-squared" acceleration of the iterates. This can improve the convergence on multiple roots where the ordinary Newton algorithm is slow.

$$x[i+1] = x[i] - f(x[i]) / f'(x[i])$$

$$x_{\text{accelerated}}[i] = x[i] - (x[i+1] - x[i])**2 / (x[i+2] - 2*x[i+1] + x[i])$$

We can only use the accelerated estimate after three iterations, and use the unaccelerated value until then.

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 $|\text{root} - x| < \text{tol}$, where $\text{tol} = \text{tolx} + \text{tolf2} \text{ root}$.

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.

Idea for future

There's some extra copying here which can probably be removed.

Idea for future

Compare directly to GSL.

Definition at line 99 of file `gsl_root_stef.h`.

Public Member Functions

- 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`
Function value.
- double `df`
Derivative value.
- double `x_1`
Previous value of root.
- double `x`
Root.
- int `count`
Number of iterations.
- func_t * `fp`
The function to solve.
- dfunc_t * `dfp`
The derivative.
- param_t * `params`
The function parameters.

8.164.2 Member Function Documentation

8.164.2.1 int iterate () [inline]

Perform an iteration.

After a successful iteration, `root` contains the most recent value of the `root`.

Definition at line 157 of file `gsl_root_stef.h`.

8.164.2.2 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 275 of file gsl_root_stef.h.

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

- gsl_root_stef.h

8.165 gsl_series Class Reference

Series acceleration by Levin u-transform (GSL).

```
#include <gsl_series.h>
```

8.165.1 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.

Idea for future

Convert to use a more general vector

Definition at line 42 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:

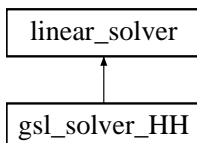
- gsl_series.h

8.166 gsl_solver_HH Class Reference

GSL Householder solver.

```
#include <linear_solver.h>
```

Inheritance diagram for gsl_solver_HH::



8.166.1 Detailed Description

GSL Householder solver.

Definition at line 172 of file linear_solver.h.

Public Member Functions

- virtual int [solve](#) (size_t n, [o2scl::omatrix_base](#) &A, [o2scl::ovector_base](#) &b, [o2scl::ovector_base](#) &x)
Solve square linear system $Ax = b$ of size n.

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

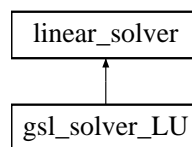
- linear_solver.h

8.167 gsl_solver_LU Class Reference

GSL solver by LU decomposition.

```
#include <linear_solver.h>
```

Inheritance diagram for `gsl_solver_LU`:



8.167.1 Detailed Description

GSL solver by LU decomposition.

Definition at line 117 of file linear_solver.h.

Public Member Functions

- virtual int [solve](#) (size_t n, [o2scl::omatrix_base](#) &A, [o2scl::ovector_base](#) &b, [o2scl::ovector_base](#) &x)
Solve square linear system $Ax = b$ of size n.

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

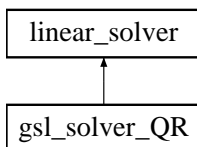
- linear_solver.h

8.168 gsl_solver_QR Class Reference

GSL solver by QR decomposition.

```
#include <linear_solver.h>
```

Inheritance diagram for `gsl_solver_QR`:



8.168.1 Detailed Description

GSL solver by QR decomposition.

Definition at line 146 of file linear_solver.h.

Public Member Functions

- virtual int [solve](#) (size_t n, [o2scl::omatrix_base](#) &A, [o2scl::ovector_base](#) &b, [o2scl::ovector_base](#) &x)
Solve square linear system $Ax = b$ of size n.

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

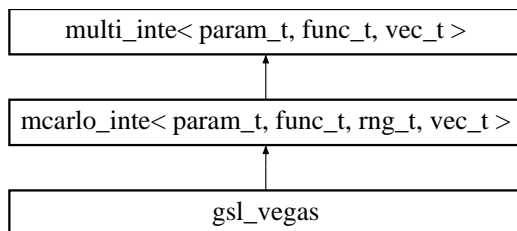
- linear_solver.h

8.169 gsl_vegas Class Template Reference

Multidimensional integration using Vegas Monte Carlo (GSL).

```
#include <gsl_vegas.h>
```

Inheritance diagram for gsl_vegas::



8.169.1 Detailed Description

```
template<class param_t, class func_t = multi_funct<param_t>, class rng_t = gsl_rnga, class vec_t = ovector_base, class
alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_vegas< param_t, func_t, rng_t, vec_t, alloc_vec_t, alloc_t >
```

Multidimensional integration using Vegas Monte Carlo (GSL).

The output options are a little different than the original GSL routine. The default setting of [mcarlo_inte::verbose](#) is 0, which turns off all output. A verbose value of 1 prints summary information about the weighted average and final result, while a value of 2 also displays the grid coordinates. A value of 3 prints information from the rebinning procedure for each iteration.

Some original documentation from GSL:

```
The input coordinates are x[j], with upper and lower limits
xu[j] and xl[j]. The integration length in the j-th direction is
delx[j]. Each coordinate x[j] is rescaled to a variable y[j] in
```

the range 0 to 1. The range is divided into bins with boundaries $xi[i][j]$, where $i=0$ corresponds to $y=0$ and $i=bins$ to $y=1$. The grid is refined (ie, bins are adjusted) using $d[i][j]$ which is some variation on the squared sum. A third parameter used in defining the real coordinate using random numbers is called z . It ranges from 0 to bins. Its integer part gives the lower index of the bin into which a call is to be placed, and the remainder gives the location inside the bin.

When stratified sampling is used the bins are grouped into boxes, and the algorithm allocates an equal number of function calls to each box.

The variable α controls how "stiff" the rebinning algorithm is. $\alpha = 0$ means never change the grid. α is typically set between 1 and 2.

Todo

Need to double check that the verbose output is good for all settings of verbose.

Todo

BINS_MAX and bins_max are somehow duplicates. Fix this.

Things to document

Document the member data more carefully

Idea for future

Could convert bins and boxes to a more useful structure

Idea for future

The testing file calls the error handler on the [composite_inte](#) section. Fix this.

Based on [Lepage78](#). The current version of the algorithm was described in the Cornell preprint CLNS-80/447 of March, 1980. The GSL code follows most closely the C version by D. R. Yennie, coded in 1984.

Definition at line 110 of file `gsl_vegas.h`.

Integration mode (default is mode_importance)

- int **mode**
- static const int **mode_importance** = 1
- static const int **mode_importance_only** = 0
- static const int **mode_stratified** = -1

Public Member Functions

- virtual int [allocate](#) (size_t ldim)
Allocate memory.
- virtual int [free](#) ()
Free allocated memory.
- virtual int [vegas_minteg_err](#) (int stage, func_t &func, size_t ndim, const vec_t &xl, const vec_t &xu, param_t &pa, double &res, double &err)
Integrate function func from $x=a$ to $x=b$.
- 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 double [minteg](#) (func_t &func, size_t ndim, const vec_t &a, const vec_t &b, param_t &pa)
Integrate function func over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i < ndim-1$.
- virtual const char * [type](#) ()
Return string denoting type ("gsl_vegas").

Data Fields

- double [result](#)
Result from last iteration.
- double [sigma](#)
Uncertainty from last iteration.
- double [alpha](#)
The stiffness of the rebinning algorithm (default 1.5).
- unsigned int [iterations](#)
Set the number of iterations (default 5).
- double [chisq](#)
The chi-squared per degree of freedom for the weighted estimate of the integral.
- std::ostream * [outs](#)
The output stream to send output information (default std::cout).

Protected Member Functions

- virtual void [init_box_coord](#) (int boxt[])
 - Initialize box coordinates.*
- int [change_box_coord](#) (int boxt[])
 - Change box coordinates.*
- virtual void [init_grid](#) (const vec_t &xl, const vec_t &xu, size_t ldim)
 - Initialize grid.*
- virtual void [reset_grid_values](#) ()
 - Reset grid values.*
- void [accumulate_distribution](#) (int lbin[], double y)
 - Add the most recently generated result to the distribution.*
- void [random_point](#) (vec_t &lx, int lbin[], double *bin_vol, const int lbox[], const vec_t &xl, const vec_t &xu)
 - Generate a random position in a given box.*
- virtual void [resize_grid](#) (unsigned int lbins)
 - Resize the grid.*
- virtual void [refine_grid](#) ()
 - Refine the grid.*
- virtual void [print_lim](#) (const vec_t &xl, const vec_t &xu, unsigned long ldim)
 - Print limits of integration.*
- virtual void [print_head](#) (unsigned long num_dim, unsigned long calls, unsigned int lit_num, unsigned int lbins, unsigned int lboxes)
 - Print header.*
- virtual void [print_res](#) (unsigned int itr, double res, double err, double cum_res, double cum_err, double chi_sq)
 - Print results.*
- virtual void [print_dist](#) (unsigned long ldim)
 - Print distribution.*
- virtual void [print_grid](#) (unsigned long ldim)
 - Print grid.*

Protected Attributes

- size_t **dim**
- size_t **bins_max**
- unsigned int **bins**
- unsigned int **boxes**

- double * **xi**
- double * **xin**
- double * **delx**
- double * **weight**
- double **vol**
- int * **bin**
- int * **box**
- double * **d**
- double **jac**
- double **wtd_int_sum**
- double **sum_wgts**
- double **chi_sum**
- unsigned int **it_start**
- unsigned int **it_num**
- unsigned int **samples**
- unsigned int **calls_per_box**
- alloc_t **ao**
Memory allocation object.
- alloc_vec_t **x**
Point for function evaluation.

Static Protected Attributes

- static const int **BINS_MAX** = 50
Desc.

8.169.2 Member Function Documentation

8.169.2.1 void accumulate_distribution (int lbin[], double y) [inline, protected]

Add the most recently generated result to the distribution.

This is among the member functions that is not virtual because it is part of the innermost loop.

Definition at line 264 of file gsl_vegas.h.

8.169.2.2 int change_box_coord (int boxt[]) [inline, protected]

Change box coordinates.

Steps through the box coord like {0,0}, {0, 1}, {0, 2}, {0, 3}, {1, 0}, {1, 1}, {1, 2}, ...

This is among the member functions that is not virtual because it is part of the innermost loop.

Definition at line 211 of file gsl_vegas.h.

8.169.2.3 void random_point (vec_t & lx, int lbin[], double * bin_vol, const int lbox[], const vec_t & xl, const vec_t & xu) [inline, protected]

Generate a random position in a given box.

Use the random number generator r to return a random position x in a given box. The value of bin gives the bin location of the random position (there may be several bins within a given box)

This is among the member functions that is not virtual because it is part of the innermost loop.

Definition at line 284 of file gsl_vegas.h.

8.169.2.4 virtual int vegas_minteg_err (int stage, func_t &func, size_t ndim, const vec_t &xl, const vec_t &xu, param_t &pa, double &res, double &err) [inline, virtual]

Integrate function `func` from `x=a` to `x=b`.

Original documentation from GSL:

Normally, 'stage = 0' which begins with a new uniform grid and empty weighted average. Calling vegas with 'stage = 1' retains the grid from the previous run but discards the weighted average, so that one can "tune" the grid using a relatively small number of points and then do a large run with 'stage = 1' on the optimized grid. Setting 'stage = 2' keeps the grid and the weighted average from the previous run, but may increase (or decrease) the number of histogram bins in the grid depending on the number of calls available. Choosing 'stage = 3' enters at the main loop, so that nothing is changed, and is equivalent to performing additional iterations in a previous call.

Definition at line 652 of file `gsl_vegas.h`.

8.169.3 Field Documentation

8.169.3.1 double alpha

The stiffness of the rebinning algorithm (default 1.5).

This usual range is between 1 and 2.

Definition at line 134 of file `gsl_vegas.h`.

8.169.3.2 double chisq

The chi-squared per degree of freedom for the weighted estimate of the integral.

From GSL documentation:

The value of CHISQ should be close to 1. A value of CHISQ which differs significantly from 1 indicates that the values from different iterations are inconsistent. In this case the weighted error will be under-estimated, and further iterations of the algorithm are needed to obtain reliable results.

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

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

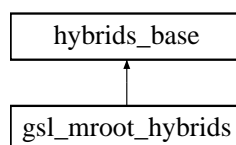
- `gsl_vegas.h`

8.170 hybrids_base Class Reference

Base functions for [gsl_mroot_hybrids](#).

```
#include <gsl_mroot_hybrids_b.h>
```

Inheritance diagram for `hybrids_base`:



8.170.1 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.

Things to document

Document the individual functions for this class, and possibly rename it.

Definition at line 46 of file `gsl_mroot_hybrids_b.h`.

Protected Member Functions

- double [enorm](#) (const `gsl_vector` *f)
Compute the norm of \vec{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 $\vec{a} + \vec{b}$.
- 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 $Q^T f$.
- 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.

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

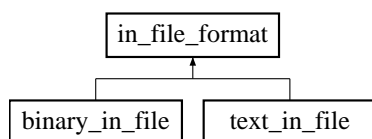
- `gsl_mroot_hybrids_b.h`

8.171 in_file_format Class Reference

Class for input file formats [abstract base].

```
#include <file_format.h>
```

Inheritance diagram for `in_file_format`:



8.171.1 Detailed Description

Class for input file formats [abstract base].

This class is experimental.

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

Public Member Functions

- 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:

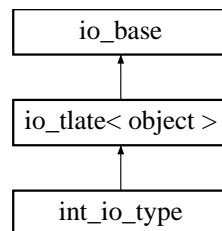
- file_format.h

8.172 int_io_type Class Reference

I/O object for int variables.

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

Inheritance diagram for int_io_type::



8.172.1 Detailed Description

I/O object for int variables.

This class is experimental.

Definition at line 2314 of file collection.h.

Public Member Functions

- [int_io_type](#) (const char *t)
Desc.

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

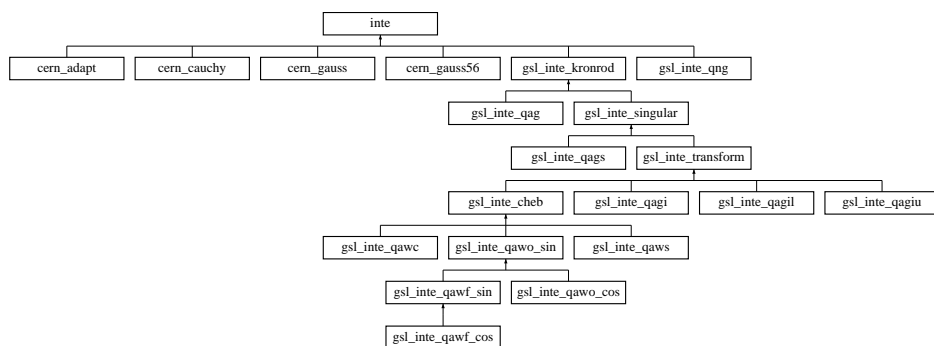
- [collection.h](#)

8.173 inte Class Template Reference

Base integration class [abstract base].

```
#include <inte.h>
```

Inheritance diagram for inte::



8.173.1 Detailed Description

```
template<class param_t, class func_t = funct<param_t>> class inte< param_t, func_t >
```

Base integration class [abstract base].

Definition at line 36 of file inte.h.

Public Member Functions

- virtual double [integ](#) (func_t &func, double a, double b, param_t &pa)=0
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)=0
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.
- int [last_iter](#)
The most recent number of iterations taken.

- 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}).
- bool [err_nonconv](#)
If true, call the error handler if the routine does not "converge".
- int [last_conv](#)
Zero if last call to [integ\(\)](#) or [integ_err\(\)](#) converged.

Protected Attributes

- double [interror](#)
The uncertainty for the last integration computation.

8.173.2 Member Function Documentation

8.173.2.1 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 94 of file inte.h.

8.173.3 Field Documentation

8.173.3.1 int last_conv

Zero if last call to [integ\(\)](#) or [integ_err\(\)](#) converged.

This is particularly useful if `err_nonconv` is false to test if the last integration call converged.

Definition at line 76 of file inte.h.

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

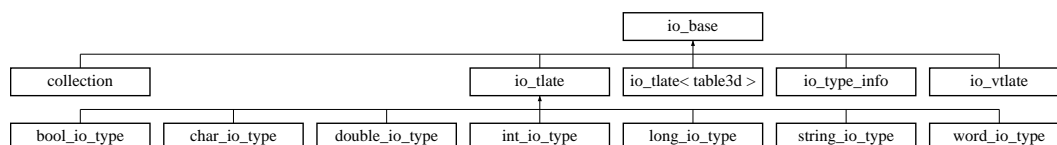
- inte.h

8.174 io_base Class Reference

I/O base class.

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

Inheritance diagram for `io_base::`



8.174.1 Detailed Description

I/O base class.

This class is experimental.

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 127 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.

8.174.2 Constructor & Destructor Documentation

8.174.2.1 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.

8.174.3 Member Function Documentation

8.174.3.1 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](#)

8.175 io_manager Class Reference

Manage I/O type information.

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

8.175.1 Detailed Description

Manage I/O type information.

This class is experimental.

This class is automatically created, utilized, and destroyed by [io_base](#). It's sole constructor is protected, so it cannot be created by a generic end-user.

Definition at line 288 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.

8.175.2 Member Function Documentation

8.175.2.1 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:

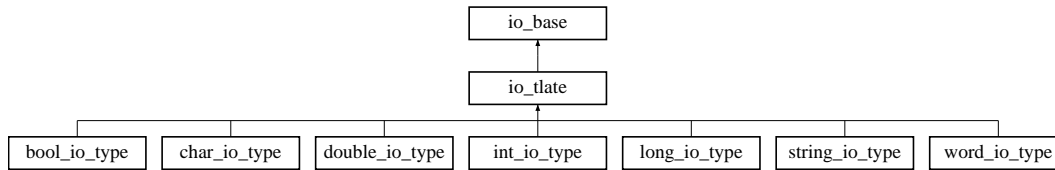
- [collection.h](#)

8.176 io_tlate Class Template Reference

A template for adding I/O classes.

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

Inheritance diagram for io_tlate::



8.176.1 Detailed Description

template<class object> class io_tlate< object >

A template for adding I/O classes.

This class is experimental.

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 1619 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, o2scl::in_file_format *ins, table_units *ta)`
- `template<>`
`int output (coutput *co, o2scl::out_file_format *outs, table_units *at)`
- `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.
- 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.

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.

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.

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)
Desc.
- virtual int `object_in_void` (`cinput` *cin, `in_file_format` *ins, void *op, int sz, std::string &name)
Desc.
- virtual int `object_in_void` (`cinput` *cin, `in_file_format` *ins, void *op, int sz, int sz2, std::string &name)
Desc.
- virtual int `object_in_mem_void` (`cinput` *cin, `in_file_format` *ins, void *&vp, std::string &name)
Desc.
- virtual int `object_in_mem_void` (`cinput` *cin, `in_file_format` *ins, void *&vp, int &sz, std::string &name)
Desc.
- virtual int `object_in_mem_void` (`cinput` *cin, `in_file_format` *ins, void *&vp, int &sz, int &sz2, std::string &name)
Desc.
- virtual int `object_out_void` (`coutput` *cout, `out_file_format` *outs, void *op, int sz=0, int sz2=0, std::string name="")
Desc.

- virtual int `stat_in_noobj` (`cinput` *cin, `in_file_format` *ins)
Desc.
- virtual int `stat_out_noobj` (`coutput` *cout, `out_file_format` *outs)
Desc.
- int `in_wrapper` (`cinput` *cin, `in_file_format` *ins, void *&vp)
Desc.
- int `in_wrapper` (`cinput` *cin, `in_file_format` *ins, void *&vp, int &sz)
Desc.
- int `in_wrapper` (`cinput` *cin, `in_file_format` *ins, void *&vp, int &sz, int &sz2)
Desc.
- int `out_wrapper` (`coutput` *cout, `out_file_format` *outs, void *vp, int sz, int sz2)
Desc.
- 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.

8.176.2 Member Function Documentation

8.176.2.1 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 1692 of file `collection.h`.

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

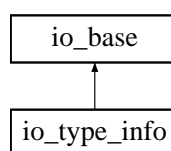
- [collection.h](#)

8.177 io_type_info Class Reference

User interface to provide I/O type information.

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

Inheritance diagram for `io_type_info`:



8.177.1 Detailed Description

User interface to provide I/O type information.

This class is experimental.

Definition at line 355 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.

8.177.2 Member Function Documentation

8.177.2.1 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.

8.177.2.2 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.

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

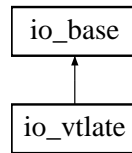
- [collection.h](#)

8.178 io_vtlate Class Template Reference

A template for adding I/O classes.

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

Inheritance diagram for io_vtlate::



8.178.1 Detailed Description

template<class object> class io_vtlate< object >

A template for adding I/O classes.

This class is experimental.

Definition at line 1542 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.

8.178.2 Member Function Documentation

8.178.2.1 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 1596 of file collection.h.

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

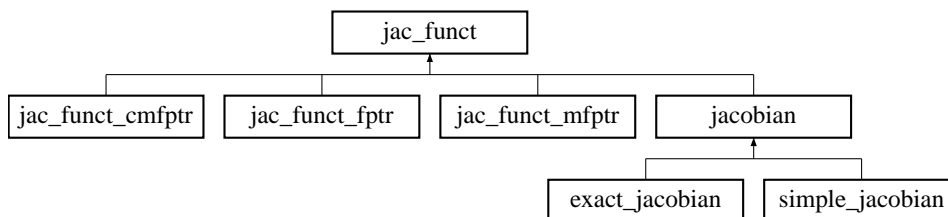
- [collection.h](#)

8.179 `jac_func` Class Template Reference

Base for a square Jacobian where J is computed at x given y=f(x) [abstract base].

```
#include <jacobian.h>
```

Inheritance diagram for `jac_func`::



8.179.1 Detailed Description

```
template<class param_t, class vec_t = ovector_base, class mat_t = omatrix_base> class jac_func< param_t, vec_t, mat_t >
```

Base for a square Jacobian where J is computed at x given y=f(x) [abstract base].

Compute

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

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 typically restored to their original values before `operator()` exits.

For Jacobian functions with C-style arrays and matrices, use the corresponding children of [jac_vfunct](#).

Definition at line 56 of file `jacobian.h`.

Public Member Functions

- virtual int [operator\(\)](#) (size_t nv, vec_t &x, vec_t &y, mat_t &j, param_t &pa)=0
The `operator()`.

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

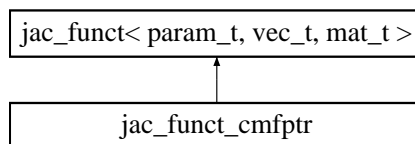
- `jacobian.h`

8.180 `jac_func_cmfp_ptr` Class Template Reference

Const member function pointer to a Jacobian.

```
#include <jacobian.h>
```

Inheritance diagram for `jac_func_cmfp_ptr`::



8.180.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base, class mat_t = omatrix_base> class jac_funct_cmfptra< tclass, param_t, vec_t, mat_t >

Const member function pointer to a Jacobian.

Definition at line 176 of file jacobian.h.

Public Member Functions

- [jac_funct_cmfptra](#) (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &y, mat_t &j, param_t &pa) const)
Specify the member function pointer.
- 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) const
Member function pointer.
- tclass * [tptr](#)
Class pointer.

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

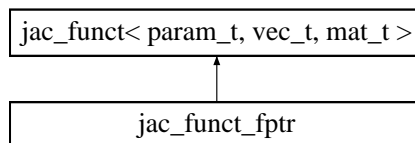
- jacobian.h

8.181 jac_funct_fptr Class Template Reference

Function pointer to [jacobian](#).

```
#include <jacobian.h>
```

Inheritance diagram for jac_funct_fptr::



8.181.1 Detailed Description

template<class param_t, class vec_t = ovector_base, class mat_t = omatrix_base> class jac_funct_fptr< param_t, vec_t, mat_t >

Function pointer to [jacobian](#).

Definition at line 83 of file jacobian.h.

Public Member Functions

- [jac_funcnt_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:

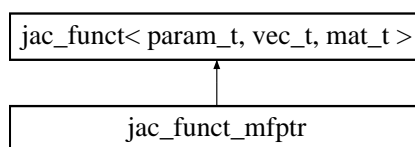
- jacobian.h

8.182 jac_funcnt_mfptr Class Template Reference

Member function pointer to a Jacobian.

```
#include <jacobian.h>
```

Inheritance diagram for jac_funcnt_mfptr::



8.182.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_base, class mat_t = omatrix_base> class jac_funcnt_mfptr< tclass,
param_t, vec_t, mat_t >
```

Member function pointer to a Jacobian.

Definition at line 126 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 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)
Member function pointer.
- tclass * [tptr](#)
Class pointer.

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

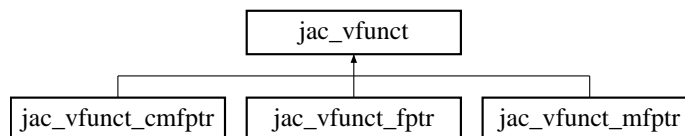
- jacobian.h

8.183 jac_vfunct Class Template Reference

Base for a square Jacobian where J is computed at x given y=f(x) with arrays [abstract base].

```
#include <jacobian.h>
```

Inheritance diagram for jac_vfunct::



8.183.1 Detailed Description

```
template<class param_t, size_t nv> class jac_vfunct< param_t, nv >
```

Base for a square Jacobian where J is computed at x given y=f(x) with arrays [abstract base].

Compute

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

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 238 of file `jacobian.h`.

Public Member Functions

- virtual int [operator\(\)](#) (size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa)=0
The `operator()`.

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

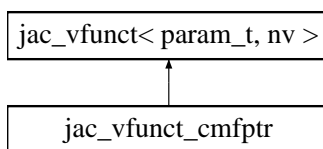
- `jacobian.h`

8.184 jac_vfunct_cmfp_ptr Class Template Reference

Const member function pointer to a Jacobian with arrays.

```
#include <jacobian.h>
```

Inheritance diagram for jac_vfunct_cmfp_ptr::



8.184.1 Detailed Description

template<class tclass, class param_t, size_t nv> class jac_vfunct_cmfp_ptr< tclass, param_t, nv >

Const member function pointer to a Jacobian with arrays.

Definition at line 355 of file jacobian.h.

Public Member Functions

- [jac_vfunct_cmfp_ptr](#) (tclass *tp, int(tclass::*fp)(size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa) const)
Specify the member function pointer.
- virtual int [operator\(\)](#) (size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa)
The operator().

Protected Attributes

- int(tclass::* [fptr](#))(size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa) const
Member function pointer.
- tclass * [tptr](#)
Class pointer.

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

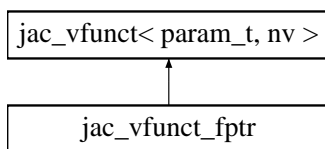
- jacobian.h

8.185 jac_vfunct_fptr Class Template Reference

Function pointer to [jacobian](#) with arrays.

```
#include <jacobian.h>
```

Inheritance diagram for jac_vfunct_fptr::



8.185.1 Detailed Description

template<class param_t, size_t nv> class jac_vfunct_fptr< param_t, nv >

Function pointer to [jacobian](#) with arrays.

Definition at line 264 of file jacobian.h.

Public Member Functions

- [jac_vfunct_fptr](#) (int(*fp)(size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa))
Specify the function pointer.
- virtual int [operator\(\)](#) (size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa)

The operator().

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

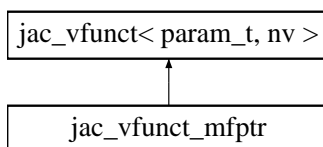
- jacobian.h

8.186 jac_vfunct_mfptr Class Template Reference

Member function pointer to a Jacobian with arrays.

```
#include <jacobian.h>
```

Inheritance diagram for jac_vfunct_mfptr::



8.186.1 Detailed Description

```
template<class tclass, class param_t, size_t nv> class jac_vfunct_mfptr< tclass, param_t, nv >
```

Member function pointer to a Jacobian with arrays.

Definition at line 306 of file jacobian.h.

Public Member Functions

- [jac_vfunct_mfptr](#) (tclass *tp, int(tclass::*fp)(size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa))
Specify the member function pointer.
- virtual int [operator\(\)](#) (size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa)
The operator().

Protected Attributes

- int(tclass::* [fptr](#))(size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa)
Member function pointer.
- tclass * [tptr](#)
Class pointer.

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

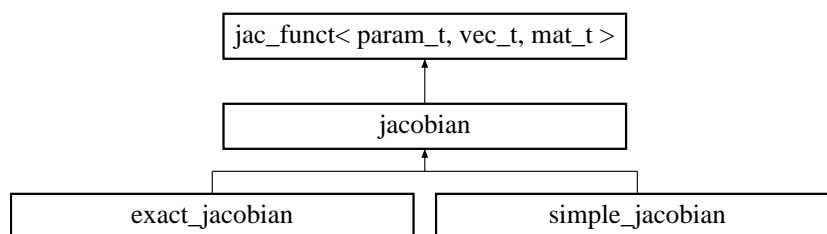
- jacobian.h

8.187 jacobian Class Template Reference

Base for providing a numerical [jacobian](#) [abstract base].

```
#include <jacobian.h>
```

Inheritance diagram for jacobian::



8.187.1 Detailed Description

template<class param_t, class func_t = mm_funct<param_t>, class vec_t = ovector_base, class mat_t = omatrix_base> class jacobian< param_t, func_t, vec_t, mat_t >

Base for providing a numerical [jacobian](#) [abstract base].

This provides a Jacobian which is numerically determined by differentiating a user-specified function (typically of the form of [mm_funct](#)).

Definition at line 411 of file jacobian.h.

Public Member Functions

- 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)=0
Evaluate the Jacobian j at point $y(x)$.

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

8.188 lanczos Class Template Reference

Lanczos diagonalization.

```
#include <lanczos_base.h>
```

8.188.1 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.

Idea for future

Do something better than the naive matrix-vector product?

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

Public Member Functions

- `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.

8.188.2 Member Function Documentation

8.188.2.1 `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 163 of file `lanczos_base.h`.

8.188.2.2 `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 `/c 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 78 of file `lanczos_base.h`.

8.188.2.3 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 300 of file `lanczos_base.h`.

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

- `lanczos_base.h`

8.189 lib_settings_class Class Reference

A class to manage global library settings.

```
#include <lib_settings.h>
```

8.189.1 Detailed Description

A class to manage global library settings.

A global object of this type is defined in `lib_settings.h` named `lib_settings`.

Definition at line 91 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.
- `bool range_check ()`
Return true if range checking was turned on during installation (default true).
- `std::string o2scl_version ()`
Return the library version.

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:

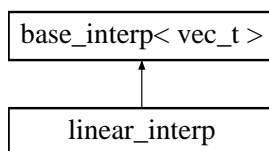
- [lib_settings.h](#)

8.190 linear_interp Class Template Reference

Linear interpolation (GSL).

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

Inheritance diagram for linear_interp::



8.190.1 Detailed Description

```
template<class vec_t> class linear_interp< vec_t >
```

Linear interpolation (GSL).

Linear interpolation requires no calls to [allocate\(\)](#), [free\(\)](#) or [init\(\)](#), as there is no internal storage required.

Definition at line 162 of file [interp.h](#).

Public Member Functions

- 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 $\int_a^b y(x) dx$.
- virtual const char * [type](#) ()
Return the type, "linear_interp".

Private Member Functions

- [linear_interp](#) (const [linear_interp](#)< vec_t > &)
- [linear_interp](#)< vec_t > & [operator=](#) (const [linear_interp](#)< vec_t > &)

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

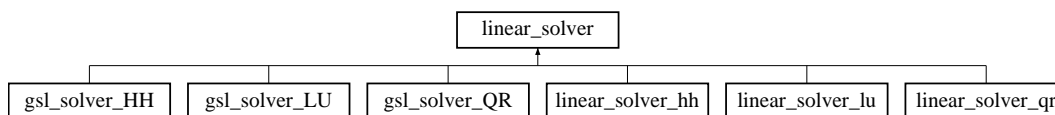
- [interp.h](#)

8.191 linear_solver Class Template Reference

A generic solver for the linear system $Ax = b$ [abstract base].

```
#include <linear_solver.h>
```


Inheritance diagram for linear_solver::



8.191.1 Detailed Description

template<class vec_t, class mat_t> class o2scl_linalg::linear_solver< vec_t, mat_t >

A generic solver for the linear system $Ax = b$ [abstract base].

A generic solver for dense linear systems.

Those writing production level code should consider calling LAPACK directly using O₂scl objects as described in the [Linear algebra](#) section of the User's Guide.

Idea for future

The test code uses a Hilbert matrix, which is known to be ill-conditioned, especially for the larger sizes. This should probably be changed.

Definition at line 50 of file linear_solver.h.

Public Member Functions

- virtual int [solve](#) (size_t n, mat_t &a, vec_t &b, vec_t &x)=0
Solve square linear system $Ax = b$ of size n.

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

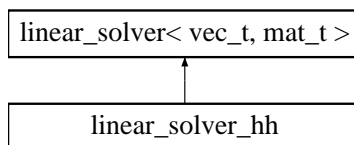
- linear_solver.h

8.192 linear_solver_hh Class Template Reference

Generic Householder linear solver.

```
#include <linear_solver.h>
```

Inheritance diagram for linear_solver_hh::



8.192.1 Detailed Description

template<class vec_t, class mat_t> class o2scl_linalg::linear_solver_hh< vec_t, mat_t >

Generic Householder linear solver.

Definition at line 100 of file linear_solver.h.

Public Member Functions

- virtual int [solve](#) (size_t n, mat_t &A, vec_t &b, vec_t &x)
Solve square linear system $Ax = b$ of size n.

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

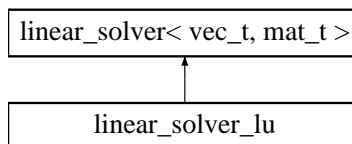
- linear_solver.h

8.193 linear_solver_lu Class Template Reference

Generic linear solver using LU decomposition.

```
#include <linear_solver.h>
```

Inheritance diagram for linear_solver_lu::



8.193.1 Detailed Description

```
template<class vec_t, class mat_t> class o2scl_linalg::linear_solver_lu< vec_t, mat_t >
```

Generic linear solver using LU decomposition.

Definition at line 60 of file linear_solver.h.

Public Member Functions

- virtual int [solve](#) (size_t n, mat_t &A, vec_t &b, vec_t &x)
Solve square linear system $Ax = b$ of size n.

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

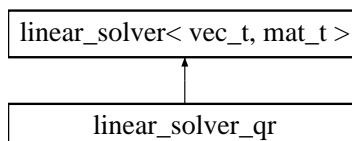
- linear_solver.h

8.194 linear_solver_qr Class Template Reference

Generic linear solver using QR decomposition.

```
#include <linear_solver.h>
```

Inheritance diagram for linear_solver_qr::



8.194.1 Detailed Description

`template<class vec_t, class mat_t> class o2scl_linalg::linear_solver_qr< vec_t, mat_t >`

Generic linear solver using QR decomposition.

Definition at line 80 of file `linear_solver.h`.

Public Member Functions

- virtual int [solve](#) (size_t n, mat_t &A, vec_t &b, vec_t &x)
Solve square linear system $Ax = b$ of size n.

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

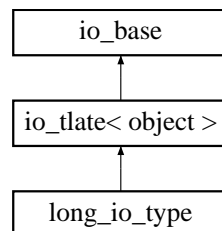
- `linear_solver.h`

8.195 `long_io_type` Class Reference

I/O object for long variables.

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

Inheritance diagram for `long_io_type`:



8.195.1 Detailed Description

I/O object for long variables.

This class is experimental.

Definition at line 2339 of file `collection.h`.

Public Member Functions

- [long_io_type](#) (const char *t)
Desc.

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

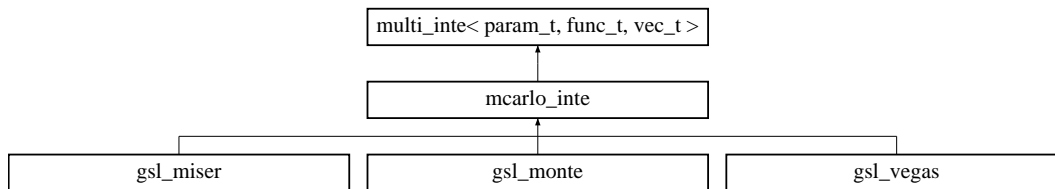
- [collection.h](#)

8.196 mcarlo_inte Class Template Reference

Monte-Carlo integration [abstract base].

```
#include <mcarlo_inte.h>
```

Inheritance diagram for mcarlo_inte::



8.196.1 Detailed Description

```
template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_base> class mcarlo_inte< param_t,
func_t, rng_t, vec_t >
```

Monte-Carlo integration [abstract base].

This class 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 44 of file mcarlo_inte.h.

Public Member Functions

- virtual const char * [type](#) ()
Return string denoting type ("mcarlo_inte").

Data Fields

- int [n_points](#)
Number of integration points (default 1000).
- rng_t [def_rng](#)
The random number generator.

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

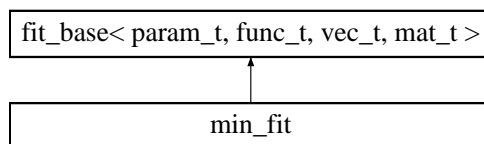
- mcarlo_inte.h

8.197 min_fit Class Template Reference

Non-linear least-squares fitting class with generic minimizer.

```
#include <min_fit.h>
```

Inheritance diagram for min_fit::



8.197.1 Detailed Description

template<class param_t, class func_t = fit_func<param_t>, class vec_t = ovector_base, class mat_t = omatrix_base> 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.

Data Structures

- struct [func_par](#)
A structure for passing information to the GSL functions for the [min_fit](#) class.

Public Member Functions

- 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_func](#)< [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_func](#)< [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.

8.197.2 Member Function Documentation

8.197.2.1 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 χ^2 is returned in `chi2`.

Implements [fit_base](#).

Definition at line 94 of file `min_fit.h`.

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

- `min_fit.h`

8.198 min_fit::func_par Struct Reference

A structure for passing information to the GSL functions for the [min_fit](#) class.

```
#include <min_fit.h>
```

8.198.1 Detailed Description

template<class [param_t](#), class [func_t](#) = [fit_func](#)<[param_t](#)>, class [vec_t](#) = [ovector_base](#), class [mat_t](#) = [omatrix_base](#)> struct [min_fit](#)< [param_t](#), [func_t](#), [vec_t](#), [mat_t](#) >::[func_par](#)

A structure for passing information to the GSL functions for the [min_fit](#) class.

This structure is given so that the user can specify the minimizer to use.

Definition at line 71 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:

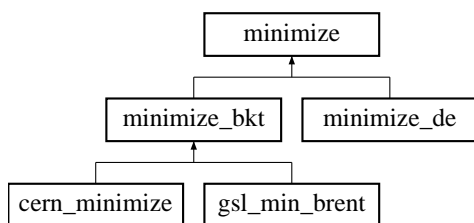
- `min_fit.h`

8.199 minimize Class Template Reference

One-dimensional minimization [abstract base].

```
#include <minimize.h>
```

Inheritance diagram for minimize::



8.199.1 Detailed Description

```
template<class param_t, class func_t, class dfunc_t = func_t> class minimize< param_t, func_t, dfunc_t >
```

One-dimensional minimization [abstract base].

Definition at line 40 of file minimize.h.

Public Member Functions

- 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)=0
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)=0
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)=0
Calculate the minimum min of func with derivative dfunc w.r.t 'x'.
- virtual int [bracket](#) (double &ax, double &bx, double &cx, double &fa, double &fb, double &fc, param_t &pa, func_t &func)
Given interval (ax, bx), attempt to 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.
- int [bracket_iter](#)
The number of iterations for automatically bracketing a minimum (default 20).
- bool [err_nonconv](#)
If true, call the error handler if the routine does not "converge".
- int [last_conv](#)
Zero if last call to [min\(\)](#), [min_bkt\(\)](#), or [min_de\(\)](#) converged.

8.199.2 Member Function Documentation

8.199.2.1 `virtual int bracket (double & ax, double & bx, double & cx, double & fa, double & fb, double & fc, param_t & pa, func_t & func)` [`inline`, `virtual`]

Given interval (ax, bx) , attempt to bracket a minimum for function `func`.

Upon success, $fa = func(ax)$, $fb = func(bx)$, and $fc = func(cx)$ with $fb < fa$, $fb < fc$ and $ax < bx < cx$. The initial values of cx , fa , fb , and fc are all ignored.

The number of iterations is controlled by [bracket_iter](#).

Note:

This algorithm can fail if there's a minimum which has a much smaller size than $bx - ax$, or if the function has the same value at ax , bx , and the midpoint $(ax + bx) / 2$.

Idea for future

Improve this algorithm with the standard golden ratio method?

Definition at line 169 of file `minimize.h`.

8.199.2.2 `virtual int min (double & x, double & fmin, param_t & pa, func_t & func)` [`pure 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.

Implemented in [minimize_bkt](#), and [minimize_de](#).

8.199.2.3 `virtual int min_bkt (double & x2, double x1, double x3, double & fmin, param_t & pa, func_t & func)` [`pure 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\(\)](#).

Implemented in [cern_minimize](#), [gsl_min_brent](#), [minimize_bkt](#), and [minimize_de](#).

8.199.2.4 `virtual int min_de (double & x, double & fmin, param_t & pa, func_t & func, dfunc_t & df)` [`pure 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.

Implemented in [minimize_bkt](#), and [minimize_de](#).

8.199.2.5 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 100 of file [minimize.h](#).

8.199.3 Field Documentation

8.199.3.1 int last_conv

Zero if last call to [min\(\)](#), [min_bkt\(\)](#), or [min_de\(\)](#) converged.

This is particularly useful if `err_nonconv` is false to test if the last call to [min\(\)](#), [min_bkt\(\)](#), or [min_de\(\)](#) converged.

Definition at line 88 of file [minimize.h](#).

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

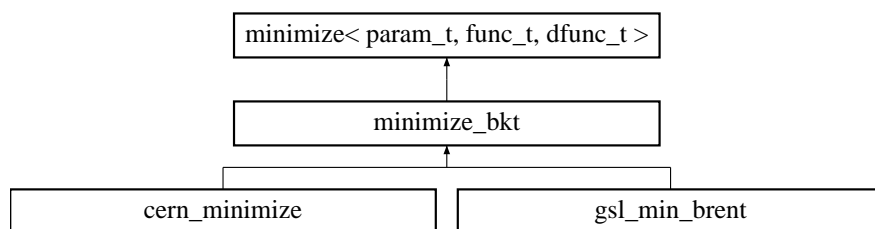
- [minimize.h](#)

8.200 minimize_bkt Class Template Reference

One-dimensional bracketing minimization [abstract base].

```
#include <minimize.h>
```

Inheritance diagram for `minimize_bkt`:



8.200.1 Detailed Description

template<class param_t, class func_t, class dfunc_t = func_t> class minimize_bkt< param_t, func_t, dfunc_t >

One-dimensional bracketing minimization [abstract base].

Definition at line 243 of file [minimize.h](#).

Public Member Functions

- 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)=0
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 const char * [type](#) ()
Return string denoting type ("minimize_bkt").

Data Fields

- int [bracket_iter](#)
The number of iterations for automatically bracketing a minimum (default 20).

8.200.2 Member Function Documentation

8.200.2.1 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.

Implements [minimize](#).

Definition at line 266 of file minimize.h.

8.200.2.2 virtual int min_bkt (double &x2, double x1, double x3, double &fmin, param_t &pa, func_t &func) [pure 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\(\)](#).

Implements [minimize](#).

Implemented in [cern_minimize](#), and [gsl_min_brent](#).

8.200.2.3 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.

Implements [minimize](#).

Definition at line 292 of file minimize.h.

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

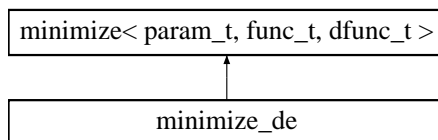
- [minimize.h](#)

8.201 minimize_de Class Template Reference

One-dimensional minimization using derivatives [abstract base].

```
#include <minimize.h>
```

Inheritance diagram for minimize_de::



8.201.1 Detailed Description

template<class param_t, class func_t, class dfunc_t = func_t> class minimize_de< param_t, func_t, dfunc_t >

One-dimensional minimization using derivatives [abstract base].

At the moment there are no minimizers of this type implemented in O₂scl .

Definition at line 316 of file minimize.h.

Public Member Functions

- 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)=0
Calculate the minimum min of func with derivative dfunc w.r.t 'x'.
- virtual const char * [type](#) ()
Return string denoting type ("minimize_de").

8.201.2 Member Function Documentation

8.201.2.1 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.

Implements [minimize](#).

Definition at line 332 of file minimize.h.

8.201.2.2 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\(\)](#).

Implements [minimize](#).

Definition at line 341 of file minimize.h.

8.201.2.3 virtual int min_de (double & x, double & fmin, param_t & pa, func_t & func, dfunc_t & df) [pure 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.

Implements [minimize](#).

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

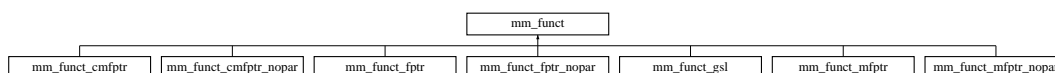
- [minimize.h](#)

8.202 mm_funct Class Template Reference

Array of multi-dimensional functions [abstract base].

```
#include <mm_funct.h>
```

Inheritance diagram for mm_funct::



8.202.1 Detailed Description

template<class param_t, class vec_t = ovector_base> class mm_funct< param_t, vec_t >

Array of multi-dimensional functions [abstract base].

This class generalizes nv functions of nv variables, i.e. $y_j(x_0, x_1, \dots, x_{nv-1})$ for $0 \leq j \leq nv - 1$.

For functions with C-style arrays, use the corresponding children of [mm_vfunct](#).

This class is one of a large number of function object classes in O₂scl designed to provide a mechanism for the user to supply functions to solvers, minimizers, integrators, etc. See [Function Objects](#) for a general description.

Definition at line 50 of file mm_funct.h.

Public Member Functions

- virtual int [operator\(\)](#) (size_t nv, const vec_t &x, vec_t &y, param_t &pa)=0
Compute nv functions, y , of nv variables stored in x with parameter pa .

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

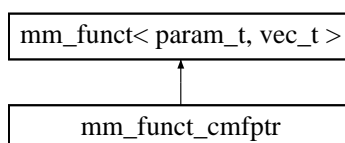
- mm_funct.h

8.203 mm_funct_cmfptr Class Template Reference

Const member function pointer to an array of multi-dimensional functions.

```
#include <mm_funct.h>
```

Inheritance diagram for mm_funct_cmfptr::



8.203.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base> class mm_funct_cmfpnr< tclass, param_t, vec_t >

Const member function pointer to an array of multi-dimensional functions.

Definition at line 335 of file mm_funct.h.

Public Member Functions

- **mm_funct_cmfpnr** ()
Empty constructor.
- **mm_funct_cmfpnr** (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, vec_t &y, param_t &pa) const)
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) const)
Specify the member 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(tclass::* **fptr**) (size_t nv, const vec_t &x, vec_t &y, param_t &pa) const
The member function pointer.
- tclass * **tptr**
The class pointer.

Private Member Functions

- **mm_funct_cmfpnr** (const **mm_funct_cmfpnr** &)
- **mm_funct_cmfpnr** & **operator=** (const **mm_funct_cmfpnr** &)

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

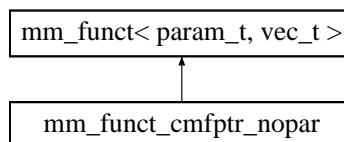
- mm_funct.h

8.204 mm_funct_cmfpnr_nopar Class Template Reference

Const member function pointer to an array of multi-dimensional functions with no parameters.

```
#include <mm_funct.h>
```

Inheritance diagram for mm_funct_cmfpnr_nopar::



8.204.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base> class mm_funct_cmfpnr_nopar< tclass, param_t, vec_t >

Const member function pointer to an array of multi-dimensional functions with no parameters.

Definition at line 393 of file mm_funct.h.

Public Member Functions

- [mm_funct_cmfp_ptr_nopar](#) ()
Empty constructor.
- [mm_funct_cmfp_ptr_nopar](#) (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, vec_t &y) const)
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 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) const
The member function pointer.
- tclass * [tptr](#)
The class pointer.

Private Member Functions

- [mm_funct_cmfp_ptr_nopar](#) (const [mm_funct_cmfp_ptr_nopar](#) &)
- [mm_funct_cmfp_ptr_nopar](#) & [operator=](#) (const [mm_funct_cmfp_ptr_nopar](#) &)

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

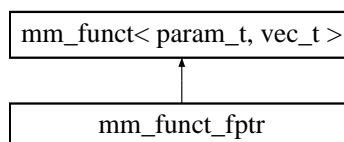
- mm_funct.h

8.205 mm_funct_fptr Class Template Reference

Function pointer to array of multi-dimensional functions.

```
#include <mm_funct.h>
```

Inheritance diagram for mm_funct_fptr::



8.205.1 Detailed Description

```
template<class param_t, class vec_t = ovector_base> class mm_funct_fptr< param_t, vec_t >
```

Function pointer to array of multi-dimensional functions.

Definition at line 77 of file mm_funct.h.

Public Member Functions

- `mm_func_ptr` (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_func_ptr` (const `mm_func_ptr` &)
- `mm_func_ptr` & `operator=` (const `mm_func_ptr` &)

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

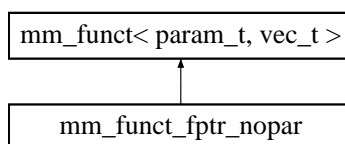
- mm_func.h

8.206 mm_func_ptr_nopar Class Template Reference

Function pointer to array of multi-dimensional functions with no parameters.

```
#include <mm_func.h>
```

Inheritance diagram for mm_func_ptr_nopar::



8.206.1 Detailed Description

```
template<class param_t, class vec_t = ovector_base> class mm_func_ptr_nopar< param_t, vec_t >
```

Function pointer to array of multi-dimensional functions with no parameters.

Definition at line 128 of file mm_func.h.

Public Member Functions

- `mm_func_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_fptr_nopar` (const `mm_funct_fptr_nopar` &)
- `mm_funct_fptr_nopar` & `operator=` (const `mm_funct_fptr_nopar` &)

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

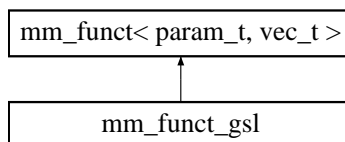
- `mm_funct.h`

8.207 mm_funct_gsl Class Template Reference

Function pointer to a `gsl_multiroot_function`.

```
#include <mm_funct.h>
```

Inheritance diagram for `mm_funct_gsl`:



8.207.1 Detailed Description

```
template<class param_t, class vec_t = ovector_base> 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_base` class because `ovector_base` 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:

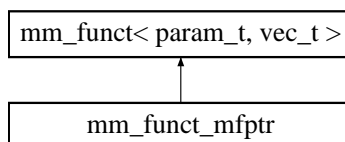
- `mm_funct.h`

8.208 mm_funct_mfptr Class Template Reference

Member function pointer to an array of multi-dimensional functions.

```
#include <mm_funct.h>
```

Inheritance diagram for `mm_funct_mfptr`:



8.208.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_base> class mm_funct_mfptr< tclass, param_t, vec_t >
```

Member function pointer to an array of multi-dimensional functions.

Definition at line 219 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 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:

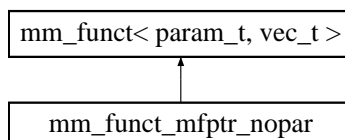
- mm_funct.h

8.209 mm_funct_mfptr_nopar Class Template Reference

Member function pointer to an array of multi-dimensional functions with no parameters.

```
#include <mm_funct.h>
```

Inheritance diagram for mm_funct_mfptr_nopar::



8.209.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_base> class mm_funct_mfptr_nopar< tclass, param_t, vec_t >
```

Member function pointer to an array of multi-dimensional functions with no parameters.

Definition at line 277 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 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:

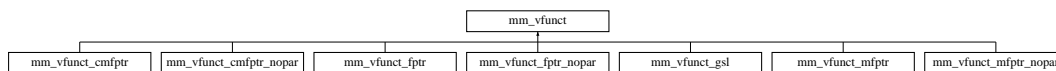
- mm_funct.h

8.210 mm_vfunct Class Template Reference

Array of multi-dimensional functions with arrays [abstract base].

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct::



8.210.1 Detailed Description

template<class param_t, size_t nv> class mm_vfunct< param_t, nv >

Array of multi-dimensional functions with arrays [abstract base].

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_base objects instead of C-style arrays, use [mm_funct](#).

This class is one of a large number of function object classes in O₂scl designed to provide a mechanism for the user to supply functions to solvers, minimizers, integrators, etc. See [Function Objects](#) for a general description.

Definition at line 464 of file mm_funct.h.

Public Member Functions

- virtual int [operator\(\)](#) (size_t nvar, const double x[nv], double y[nv], param_t &pa)=0
Compute nv functions, y, of nv variables stored in x with parameter pa.

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

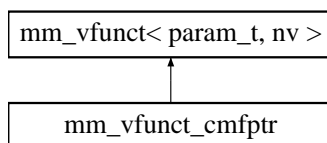
- mm_funct.h

8.211 mm_vfunct_cmfprr Class Template Reference

Const member function pointer to an array of multi-dimensional functions with arrays.

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct_cmfprr::



8.211.1 Detailed Description

template<class tclass, class param_t, size_t nv> class mm_vfunct_cmfprr< tclass, param_t, nv >

Const member function pointer to an array of multi-dimensional functions with arrays.

Definition at line 711 of file mm_funct.h.

Public Member Functions

- [mm_vfunct_cmfp_ptr](#) (tclass *tp, int(tclass::*fp)(size_t nvar, const double x[nv], double y[nv], param_t &pa) const)
Specify the member 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(tclass::* [fp_ptr](#)) (size_t nvar, const double x[nv], double y[nv], param_t &pa) const
The member function pointer.
- tclass * [tp_ptr](#)
The class pointer.

Private Member Functions

- [mm_vfunct_cmfp_ptr](#) (const [mm_vfunct_cmfp_ptr](#) &)
- [mm_vfunct_cmfp_ptr](#) & [operator=](#) (const [mm_vfunct_cmfp_ptr](#) &)

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

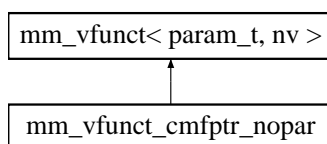
- mm_funct.h

8.212 mm_vfunct_cmfp_ptr_nopar Class Template Reference

Member function pointer to an array of multi-dimensional functions with arrays with no parameters.

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct_cmfp_ptr_nopar::



8.212.1 Detailed Description

```
template<class tclass, class param_t, size_t nv> class mm_vfunct_cmfp_ptr_nopar< tclass, param_t, nv >
```

Member function pointer to an array of multi-dimensional functions with arrays with no parameters.

Definition at line 757 of file mm_funct.h.

Public Member Functions

- [mm_vfunct_cmfp_ptr_nopar](#) (tclass *tp, int(tclass::*fp)(size_t nvar, const double x[nv], double y[nv]) const)
Specify the member 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(tclass::* fptr)(size_t nvar, const double x[nv], double y[nv]) const`
The member function pointer.
- `tclass * tptr`
The class pointer.

Private Member Functions

- `mm_vfunct_cmfp_ptr_nopar(const mm_vfunct_cmfp_ptr_nopar &)`
- `mm_vfunct_cmfp_ptr_nopar & operator=(const mm_vfunct_cmfp_ptr_nopar &)`

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

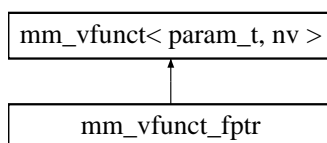
- `mm_funct.h`

8.213 mm_vfunct_fptr Class Template Reference

Function pointer to array of multi-dimensional functions with arrays.

```
#include <mm_funct.h>
```

Inheritance diagram for `mm_vfunct_fptr`:



8.213.1 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 491 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 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)(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:

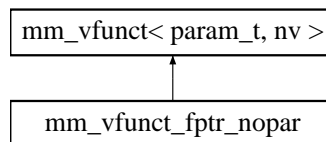
- `mm_func.h`

8.214 mm_vfunct_fptr_nopar Class Template Reference

Function pointer to array of multi-dimensional functions with arrays and no parameters.

```
#include <mm_func.h>
```

Inheritance diagram for `mm_vfunct_fptr_nopar`:



8.214.1 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 535 of file `mm_func.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 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`)(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:

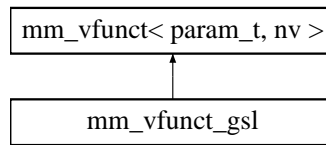
- `mm_func.h`

8.215 mm_vfunct_gsl Class Template Reference

Function pointer to a gsl_multiroot_function with arrays.

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct_gsl::



8.215.1 Detailed Description

```
template<class param_t, size_t nv> class mm_vfunct_gsl< param_t, nv >
```

Function pointer to a gsl_multiroot_function with arrays.

Definition at line 580 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:

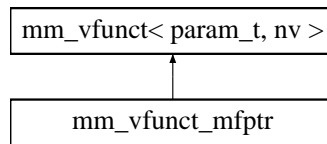
- mm_funct.h

8.216 mm_vfunct_mfptr Class Template Reference

Member function pointer to an array of multi-dimensional functions with arrays.

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct_mfptr::



8.216.1 Detailed Description

template<class tclass, class param_t, size_t nv> class mm_vfunct_mfptr< tclass, param_t, nv >

Member function pointer to an array of multi-dimensional functions with arrays.

Definition at line 619 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 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:

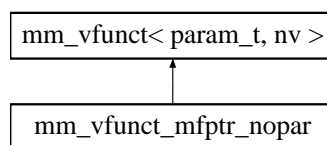
- mm_funct.h

8.217 mm_vfunct_mfptr_nopar Class Template Reference

Const member function pointer to an array of multi-dimensional functions with arrays with no parameters.

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct_mfptr_nopar::



8.217.1 Detailed Description

template<class tclass, class param_t, size_t nv> class mm_vfunct_mfptr_nopar< tclass, param_t, nv >

Const member function pointer to an array of multi-dimensional functions with arrays with no parameters.

Definition at line 665 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 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:

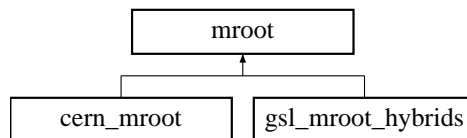
- mm_funct.h

8.218 mroot Class Template Reference

Multidimensional root-finding [abstract base].

```
#include <mroot.h>
```

Inheritance diagram for mroot::



8.218.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_base, class jfunc_t = jac_func<param_t,vec_t,omatrix_base>>
class mroot< param_t, func_t, vec_t, jfunc_t >

Multidimensional root-finding [abstract 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.

There is an example for the usage of the multidimensional solver classes given in `examples/ex_mroot.cpp`, see [Multidimensional solver example](#).

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

Public Member Functions

- virtual const char * [type](#) ()
Return the type, "mroot".
- virtual int [msolve](#) (size_t n, vec_t &x, param_t &pa, func_t &func)=0
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.
- bool [err_nonconv](#)
If true, call the error handler if [msolve\(\)](#) or [msolve_de\(\)](#) does not converge (default true).
- int [last_conv](#)
Zero if last call to [msolve\(\)](#) or [msolve_de\(\)](#) converged.

8.218.2 Member Function Documentation

8.218.2.1 `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`, and `gsl_mroot_hybrids< param_t, mm_func< param_t > >`.

Definition at line 119 of file `mroot.h`.

8.218.2.2 `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 139 of file `mroot.h`.

8.218.3 Field Documentation

8.218.3.1 `int last_conv`

Zero if last call to `msolve()` or `msolve_de()` converged.

This is particularly useful if `err_nonconv` is false to test if the last call to `msolve()` or `msolve_de()` converged.

Definition at line 105 of file `mroot.h`.

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

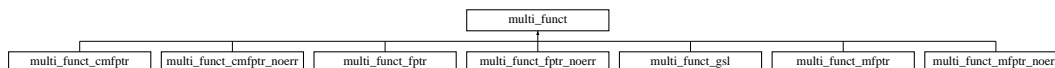
- `mroot.h`

8.219 multi_func Class Template Reference

Multi-dimensional function [abstract base].

`#include <multi_func.h>`

Inheritance diagram for `multi_func`:



8.219.1 Detailed Description

`template<class param_t, class vec_t = ovector_base> class multi_func< param_t, vec_t >`

Multi-dimensional function [abstract 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 .

For functions with C-style arrays, use the corresponding children of [multi_vfunct](#).

This class is one of a large number of function object classes in O₂scl designed to provide a mechanism for the user to supply functions to solvers, minimizers, integrators, etc. See [Function Objects](#) for a general description.

Definition at line 50 of file multi_func.h.

Public Member Functions

- virtual int [operator\(\)](#) (size_t nv, const vec_t &x, double &y, param_t &pa)=0
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 .

8.219.2 Member Function Documentation

8.219.2.1 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 in [multi_func_fptr](#), [multi_func_gsl](#), [multi_func_fptr_noerr](#), [multi_func_mfprr](#), [multi_func_mfprr_noerr](#), [multi_func_cmfprr](#), and [multi_func_cmfprr_noerr](#).

Definition at line 71 of file multi_func.h.

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

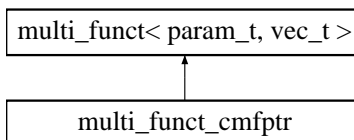
- multi_func.h

8.220 multi_func_cmfprr Class Template Reference

Const member function pointer to a multi-dimensional function.

```
#include <multi_func.h>
```

Inheritance diagram for multi_func_cmfprr:



8.220.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_base> class multi_func_cmfprr< tclass, param_t, vec_t >
```

Const member function pointer to a multi-dimensional function.

Definition at line 374 of file multi_func.h.

Public Member Functions

- [multi_funct_cmfprr](#) (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, double &y, param_t &pa) const)
Specify the member function pointer.
- 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) const
Store the function pointer.
- tclass * [tptr](#)
Store a pointer to the class instance.

Private Member Functions

- [multi_funct_cmfprr](#) (const [multi_funct_cmfprr](#) &)
- [multi_funct_cmfprr](#) & [operator=](#) (const [multi_funct_cmfprr](#) &)

8.220.2 Member Function Documentation

8.220.2.1 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 401 of file `multi_funct.h`.

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

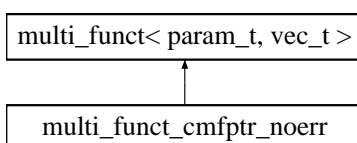
- `multi_funct.h`

8.221 multi_funct_cmfprr_noerr Class Template Reference

Const member function pointer to a multi-dimensional function.

```
#include <multi_funct.h>
```

Inheritance diagram for `multi_funct_cmfprr_noerr`:



8.221.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_base> class multi_funct_cmfprr_noerr< tclass, param_t, vec_t >
```

Const member function pointer to a multi-dimensional function.

Definition at line 432 of file multi_funct.h.

Public Member Functions

- [multi_funct_cmfp_ptr_noerr](#) (tclass *tp, double(tclass::*fp)(size_t nv, const vec_t &x, param_t &pa) const)
Specify the member function pointer.
- 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) const
Store the function pointer.
- tclass * [tptr](#)
Store a pointer to the class instance.

Private Member Functions

- [multi_funct_cmfp_ptr_noerr](#) (const [multi_funct_cmfp_ptr_noerr](#) &)
- [multi_funct_cmfp_ptr_noerr](#) & [operator=](#) (const [multi_funct_cmfp_ptr_noerr](#) &)

8.221.2 Member Function Documentation

8.221.2.1 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 459 of file multi_funct.h.

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

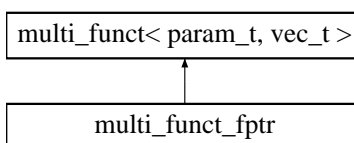
- multi_funct.h

8.222 multi_funct_fptr Class Template Reference

Function pointer to a multi-dimensional function.

```
#include <multi_funct.h>
```

Inheritance diagram for multi_funct_fptr::



8.222.1 Detailed Description

template<class param_t, class vec_t = ovector_base> class multi_funct_fptr< param_t, vec_t >

Function pointer to a multi-dimensional function.

Definition at line 91 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 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(* [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](#) &)

8.222.2 Member Function Documentation

8.222.2.1 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 118 of file multi_funct.h.

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

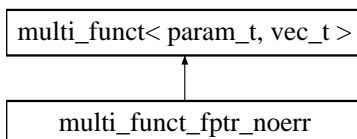
- multi_funct.h

8.223 multi_funct_fptr_noerr Class Template Reference

Function pointer to a multi-dimensional function without error control.

```
#include <multi_funct.h>
```

Inheritance diagram for multi_funct_fptr_noerr::



8.223.1 Detailed Description

template<class param_t, class vec_t = ovector_base> class multi_funct_fptr_noerr< param_t, vec_t >

Function pointer to a multi-dimensional function without error control.

Definition at line 205 of file multi_funct.h.

Public Member Functions

- [multi_funct_fptr_noerr](#) (double(*fp)(size_t nv, const vec_t &x, param_t &pa))
Specify the function pointer.
- 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(* [fptr](#))(size_t nv, const vec_t &x, param_t &pa)
Store the function pointer.

Private Member Functions

- [multi_funct_fptr_noerr](#) (const [multi_funct_fptr_noerr](#) &)
- [multi_funct_fptr_noerr](#) & [operator=](#) (const [multi_funct_fptr_noerr](#) &)

8.223.2 Member Function Documentation

8.223.2.1 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 231 of file multi_funct.h.

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

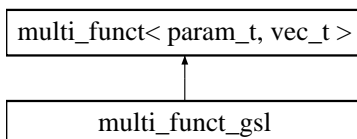
- multi_funct.h

8.224 multi_funct_gsl Class Template Reference

Function pointer to a gsl_multimin_function.

```
#include <multi_funct.h>
```

Inheritance diagram for multi_funct_gsl::



8.224.1 Detailed Description

template<class param_t, class vec_t = ovector_base> class multi_funct_gsl< param_t, vec_t >

Function pointer to a gsl_multimin_function.

Definition at line 148 of file multi_funct.h.

Public Member Functions

- [multi_funct_gsl](#) (double(*fp)(const gsl_vector *x, param_t &pa))
Specify the function pointer.
- 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(* [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](#) &)

8.224.2 Member Function Documentation

8.224.2.1 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 174 of file multi_funct.h.

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

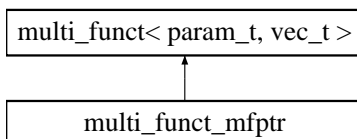
- multi_funct.h

8.225 multi_funct_mfptr Class Template Reference

Member function pointer to a multi-dimensional function.

```
#include <multi_funct.h>
```

Inheritance diagram for multi_funct_mfptr::



8.225.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base> class multi_funct_mfptr< tclass, param_t, vec_t >

Member function pointer to a multi-dimensional function.

Definition at line 261 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 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](#) &)

8.225.2 Member Function Documentation

8.225.2.1 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 287 of file multi_funct.h.

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

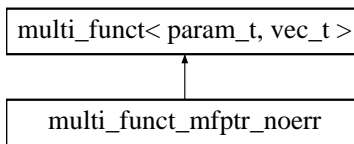
- multi_funct.h

8.226 multi_funct_mfptr_noerr Class Template Reference

Member function pointer to a multi-dimensional function.

```
#include <multi_funct.h>
```

Inheritance diagram for multi_funct_mfptr_noerr::



8.226.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base> class multi_funct_mfptr_noerr< tclass, param_t, vec_t >

Member function pointer to a multi-dimensional function.

Definition at line 317 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 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](#) &)

8.226.2 Member Function Documentation

8.226.2.1 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 344 of file multi_funct.h.

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

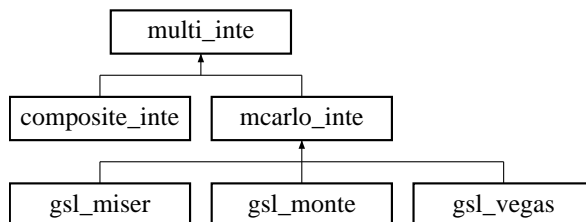
- multi_funct.h

8.227 multi_inte Class Template Reference

Multi-dimensional integration over a hypercube [abstract base].

```
#include <multi_inte.h>
```

Inheritance diagram for multi_inte::



8.227.1 Detailed Description

```
template<class param_t, class func_t, class vec_t = ovector_base> class multi_inte< param_t, func_t, vec_t >
```

Multi-dimensional integration over a hypercube [abstract base].

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 42 of file multi_inte.h.

Public Member Functions

- virtual double [minteg](#) (func_t &func, size_t ndim, const vec_t &a, const vec_t &b, param_t &pa)
Integrate function func over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i < ndim-1$.
- 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)=0
Integrate function func over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i < ndim-1$.
- 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

- bool [err_nonconv](#)
If true, call the error handler if the routine does not "converge".
- 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.

8.227.2 Member Function Documentation

8.227.2.1 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 97 of file multi_inte.h.

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

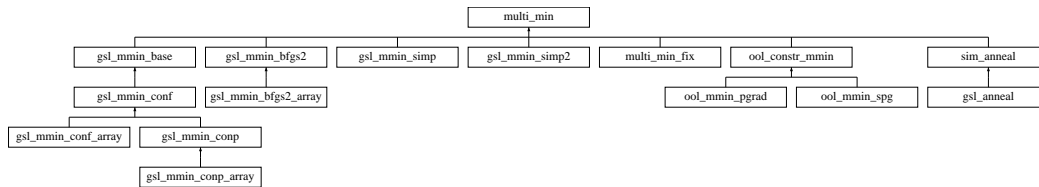
- multi_inte.h

8.228 multi_min Class Template Reference

Multidimensional minimization [abstract base].

```
#include <multi_min.h>
```

Inheritance diagram for multi_min::



8.228.1 Detailed Description

```
template<class param_t, class func_t, class dfunc_t = func_t, class vec_t = ovector_base> class multi_min< param_t, func_t, dfunc_t, vec_t >
```

Multidimensional minimization [abstract 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 538 of file multi_min.h.

Public Member Functions

- int [set_verbose_stream](#) (std::ostream &out, std::istream &in)
Set streams for verbose I/O.
- virtual int [mmin](#) (size_t nvar, vec_t &x, double &fmin, param_t &pa, func_t &func)=0
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](#)
Function value tolerance.
- double [tolx](#)
The independent variable tolerance.
- int [last_ntrial](#)
The number of iterations for in the most recent minimization.
- bool [err_nonconv](#)
If true, call the error handler if the routine does not "converge".
- int [last_conv](#)
Zero if last call to [mmin\(\)](#) or [mmin_de\(\)](#) converged.

Protected Attributes

- std::ostream * [outs](#)
Stream for verbose output.
- std::istream * [ins](#)
Stream for verbose input.

8.228.2 Member Function Documentation

8.228.2.1 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 628 of file multi_min.h.

8.228.3 Field Documentation

8.228.3.1 int last_conv

Zero if last call to [mmin\(\)](#) or [mmin_de\(\)](#) converged.

This is particularly useful if err_nonconv is false to test if the last call to [mmin\(\)](#) or [mmin_de\(\)](#) converged.

Definition at line 592 of file multi_min.h.

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

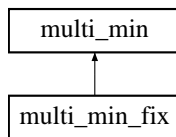
- multi_min.h

8.229 multi_min_fix Class Template Reference

Multidimensional minimizer fixing some variables and varying others.

```
#include <multi_min_fix.h>
```

Inheritance diagram for multi_min_fix::



8.229.1 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.

See an example for the usage of this class in [Minimizer fixing variables example](#) .

Todo

Generalize to all vector types

Todo

Generalize to minimizers which require derivatives

Definition at line 45 of file multi_min_fix.h.

Public Member Functions

- [multi_min_fix](#) ()
Specify the member function pointer.
- virtual int [mmin](#) (size_t nvar, [ovector_base](#) &x, double &fmin, param_t &pa, [multi_funcnt](#)< 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_base](#) &x, double &fmin, bool_vec_t &fix, param_t &pa, [multi_funcnt](#)< param_t > &func)
Calculate the minimum of func while fixing some parameters as specified in fix.
- int [set_mmin](#) ([multi_min](#)< param_t, [multi_funcnt_mfptr](#)< [multi_min_fix](#), param_t > > &min)
Change the base minimizer.

Data Fields

- [gsl_mmin_simp](#)< param_t, [multi_funcnt_mfptr](#)< [multi_min_fix](#), param_t > > [def_mmin](#)
The default base minimizer.

Protected Member Functions

- virtual int [min_func](#) (size_t nv, const [ovector_base](#) &x, double &y, param_t &pa)
The new function to send to the minimizer.

Protected Attributes

- `multi_min`< param_t, multi_funct_mfptr< multi_min_fix, param_t > > * `mmp`
The minimizer.
- `multi_funct`< 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_base` * `xp`
The user-specified initial vector.

Private Member Functions

- `multi_min_fix` (const `multi_min_fix` &)
- `multi_min_fix` & `operator=` (const `multi_min_fix` &)

8.229.2 Member Function Documentation

8.229.2.1 `virtual int mmin_fix (size_t nvar, ovector_base & x, double & fmin, bool_vec_t & fix, param_t & pa, multi_funct< param_t > & func) [inline, virtual]`

Calculate the minimum of `func` while fixing some parameters as specified in `fix`.

If all of entries `fix[0]`, `fix[1]`, ... `fix[nvar-1]` are true, then this function assumes all of the parameters are fixed and that there is no minimization to be performed. In this case, it will return 0 for success without calling the error handler.

Definition at line 107 of file `multi_min_fix.h`.

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

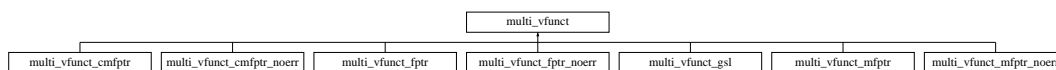
- `multi_min_fix.h`

8.230 multi_vfunct Class Template Reference

Multi-dimensional function base with arrays [abstract base].

```
#include <multi_funct.h>
```

Inheritance diagram for `multi_vfunct`:



8.230.1 Detailed Description

```
template<class param_t, size_t nvar> class multi_vfunct< param_t, nvar >
```

Multi-dimensional function base with arrays [abstract 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`.

This class is one of a large number of function object classes in O₂scl designed to provide a mechanism for the user to supply functions to solvers, minimizers, integrators, etc. See [Function Objects](#) for a general description.

Definition at line 501 of file multi_funct.h.

Public Member Functions

- virtual int [operator\(\)](#) (size_t nv, const double x[nvar], double &y, param_t &pa)=0
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 .

8.230.2 Member Function Documentation

8.230.2.1 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_mfprr](#), [multi_vfunct_mfprr_noerr](#), [multi_vfunct_cmfprr](#), and [multi_vfunct_cmfprr_noerr](#).

Definition at line 521 of file multi_funct.h.

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

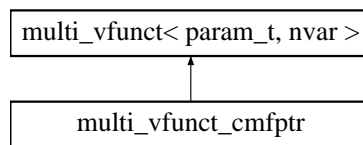
- multi_funct.h

8.231 multi_vfunct_cmfprr Class Template Reference

Member function pointer to a multi-dimensional function with arrays.

```
#include <multi_funct.h>
```

Inheritance diagram for multi_vfunct_cmfprr::



8.231.1 Detailed Description

```
template<class tclass, class param_t, size_t nvar> class multi_vfunct_cmfprr< tclass, param_t, nvar >
```

Member function pointer to a multi-dimensional function with arrays.

Definition at line 836 of file multi_funct.h.

Public Member Functions

- [multi_vfunct_cmfprr](#) (tclass *tp, int(tclass::*fp)(size_t nv, const double x[nvar], double &y, param_t &pa) const)
Specify the member function pointer.
- 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) const
Store the function pointer.
- tclass * [tptr](#)
Store a pointer to the class instance.

Private Member Functions

- [multi_vfunct_cmfprr](#) (const [multi_vfunct_cmfprr](#) &)
- [multi_vfunct_cmfprr](#) & [operator=](#) (const [multi_vfunct_cmfprr](#) &)

8.231.2 Member Function Documentation

8.231.2.1 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 864 of file `multi_funct.h`.

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

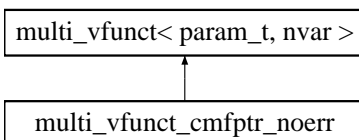
- `multi_funct.h`

8.232 multi_vfunct_cmfprr_noerr Class Template Reference

Member function pointer to a multi-dimensional function with arrays.

```
#include <multi_funct.h>
```

Inheritance diagram for `multi_vfunct_cmfprr_noerr`:



8.232.1 Detailed Description

```
template<class tclass, class param_t, size_t nvar> class multi_vfunct_cmfprr_noerr< tclass, param_t, nvar >
```

Member function pointer to a multi-dimensional function with arrays.

Definition at line 897 of file `multi_funct.h`.

Public Member Functions

- [multi_vfunct_cmfp_ptr_noerr](#) (tclass *tp, double(tclass::*fp)(size_t nv, const double x[nvar], param_t &pa) const)
Specify the member function pointer.
- 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::* [fp_ptr](#)) (size_t nv, const double x[nvar], param_t &pa) const
Store the function pointer.
- tclass * [tp_ptr](#)
Store a pointer to the class instance.

Private Member Functions

- [multi_vfunct_cmfp_ptr_noerr](#) (const [multi_vfunct_cmfp_ptr_noerr](#) &)
- [multi_vfunct_cmfp_ptr_noerr](#) & [operator=](#) (const [multi_vfunct_cmfp_ptr_noerr](#) &)

8.232.2 Member Function Documentation

8.232.2.1 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 926 of file `multi_funct.h`.

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

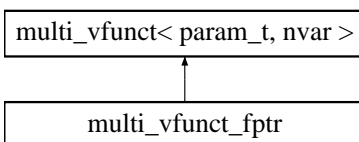
- `multi_funct.h`

8.233 multi_vfunct_fptr Class Template Reference

Function pointer to a multi-dimensional function with arrays.

```
#include <multi_funct.h>
```

Inheritance diagram for `multi_vfunct_fptr`:



8.233.1 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 541 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 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(* **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** &)

8.233.2 Member Function Documentation

8.233.2.1 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 569 of file multi_funct.h.

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

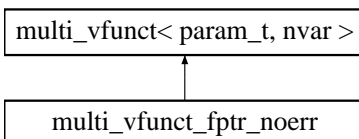
- multi_funct.h

8.234 multi_vfunct_fptr_noerr Class Template Reference

Function pointer to a multi-dimensional function with arrays and without error control.

```
#include <multi_funct.h>
```

Inheritance diagram for multi_vfunct_fptr_noerr::



8.234.1 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 657 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 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(* [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](#) &)

8.234.2 Member Function Documentation

8.234.2.1 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 684 of file multi_funct.h.

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

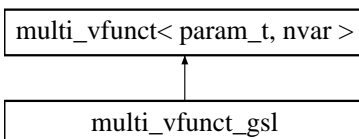
- multi_funct.h

8.235 multi_vfunct_gsl Class Template Reference

Function pointer to a gsl_multimin_function with arrays.

```
#include <multi_funct.h>
```

Inheritance diagram for multi_vfunct_gsl::



8.235.1 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 599 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 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(* [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](#) &)

8.235.2 Member Function Documentation

8.235.2.1 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 626 of file multi_funct.h.

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

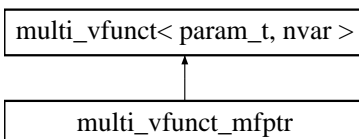
- multi_funct.h

8.236 multi_vfunct_mfp_ptr Class Template Reference

Member function pointer to a multi-dimensional function with arrays.

```
#include <multi_funct.h>
```

Inheritance diagram for multi_vfunct_mfp_ptr::



8.236.1 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 715 of file multi_funcnt.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 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](#) &)

8.236.2 Member Function Documentation

8.236.2.1 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 743 of file multi_funcnt.h.

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

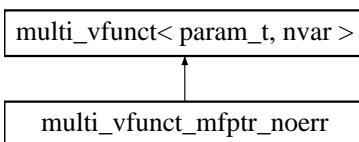
- multi_funcnt.h

8.237 multi_vfunct_mfptr_noerr Class Template Reference

Member function pointer to a multi-dimensional function with arrays.

```
#include <multi_funcnt.h>
```

Inheritance diagram for multi_vfunct_mfptr_noerr::



8.237.1 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 776 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 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](#) &)

8.237.2 Member Function Documentation

8.237.2.1 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 804 of file multi_funct.h.

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

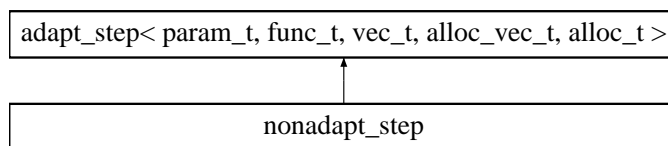
- multi_funct.h

8.238 nonadapt_step Class Template Reference

An non-adaptive stepper implementation of [adapt_step](#).

```
#include <nonadapt_step.h>
```

Inheritance diagram for nonadapt_step::



8.238.1 Detailed Description

```
template<class param_t, class func_t = ode_func<param_t>, class vec_t = ovector_base, 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](#).

This class simply calls the specified ODE stepper without any attempt to modify the size of the step, and is primarily useful to allow for simple comparisons between adaptive and non-adaptive solution. To modify the ODE stepper which is used, use the [adapt_step::set_step\(\)](#).

Idea for future

Modify so that memory allocation/deallocation is only performed when necessary

Definition at line 49 of file nonadapt_step.h.

Public Member Functions

- virtual int [astep](#) (double &x, double &h, double xmax, size_t n, vec_t &y, vec_t &u_dydx_out, vec_t &yerr, 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, vec_t &yerr, param_t &pa, func_t &derivs)
Make an adaptive integration step of the system derivs with derivatives.

Protected Attributes

- size_t [msize](#)
The allocated vector size.
- alloc_vec_t [dydx_int](#)
Internal storage for dydx.
- alloc_t [ao](#)
Memory allocator for objects of type alloc_vec_t.

8.238.2 Member Function Documentation

8.238.2.1 virtual int [astep](#) (double &x, double &h, double xmax, size_t n, vec_t &y, vec_t &u_dydx_out, vec_t &yerr, 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, `dydx_out` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

If the base stepper returns a non-zero value, that value is passed on as the return value of `astep()`, though the input y may still have been modified by the base stepper.

Implements [adapt_step](#).

Definition at line 103 of file `nonadapt_step.h`.

8.238.2.2 `virtual int astep_derivs (double & x, double & h, double xmax, size_t n, vec_t & y, vec_t & dydx, vec_t & yerr, 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, `dydx` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

If the base stepper returns a non-zero value, that value is passed on as the return value of `astep()`, though the inputs y and `dydx` may still have been modified by the base stepper.

Implements [adapt_step](#).

Definition at line 138 of file `nonadapt_step.h`.

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

- `nonadapt_step.h`

8.239 o2scl_hybrid_state_t Class Template Reference

State class for [gsl_mroot_hybrids](#).

```
#include <gsl_mroot_hybrids.h>
```

8.239.1 Detailed Description

```
template<class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base,
class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc> class o2scl_hybrid_state_t< vec_t, alloc_vec_t, alloc_t, mat_t,
alloc_mat_t, mat_alloc_t >
```

State class for [gsl_mroot_hybrids](#).

Things to document

Improve the documentation in this class, and possibly rename it.

Definition at line 67 of file `gsl_mroot_hybrids.h`.

Public Member Functions

- `int allocate (size_t n)`
Allocate memory for a solver with n variables.
- `int free ()`
Free allocated memory.

Data Fields

- `alloc_t va`
Vector allocator.
- `mat_alloc_t ma`
Matrix allocator.
- `size_t iter`
Number of iterations.
- `size_t ncfail`
Desc.
- `size_t ncsuc`
Desc.
- `size_t nslow1`
Desc.
- `size_t nslow2`
Desc.
- `double fnorm`
Desc.
- `double delta`
Desc.
- `alloc_mat_t J`
Jacobian.
- `gsl_matrix * q`
Q matrix from QR decomposition.
- `gsl_matrix * r`
R matrix from QR decomposition.
- `gsl_vector * tau`
tau vector from QR decomposition
- `gsl_vector * diag`
Desc.
- `gsl_vector * qtf`
Desc.
- `gsl_vector * newton`
Desc.
- `gsl_vector * gradient`
Desc.
- `gsl_vector * df`
Desc.
- `gsl_vector * qtdf`
Desc.
- `gsl_vector * rdx`
Desc.
- `gsl_vector * w`
Desc.
- `gsl_vector * v`
Desc.
- `size_t dim2`
Number of variables.

8.239.2 Member Function Documentation

8.239.2.1 `int allocate(size_t n)` [inline]

Allocate memory for a solver with `n` variables.

Idea for future

Convert to using `gsl_alloc_arrays()`

Definition at line 79 of file gsl_mroot_hybrids.h.

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

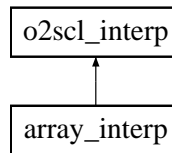
- `gsl_mroot_hybrids.h`

8.240 o2scl_interp Class Template Reference

Interpolation class.

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

Inheritance diagram for o2scl_interp::



8.240.1 Detailed Description

```
template<class vec_t = ovector_const_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 type of interpolation to be performed can be specified using the [set_type\(\)](#) function or in the constructor. The default is cubic splines with natural boundary conditions.

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.

Todo

Make sure the min size is checked on both 'itp' and 'ritp'.

Definition at line 1097 of file `interp.h`.

Public Member Functions

- [o2scl_interp](#) ([base_interp_mgr](#)< `vec_t` > &it, [base_interp_mgr](#)< `rvec_t` > &rit)
Create with base interpolation objects it and rit.
- [o2scl_interp](#) ()
Create an interpolator using the default cubic spline interpolation.
- 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_mgr< vec_t > &it, base_interp_mgr< rvec_t > &rit)
Set base interpolation object.

Data Fields

- [def_interp_mgr](#)< vec_t, cspline_interp > [dim1](#)
Default base interpolation object (cubic spline with natural boundary conditions).
- [def_interp_mgr](#)< rvec_t, cspline_interp > [dim2](#)
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.
- [base_interp_mgr](#)< vec_t > * [bim1](#)
Pointer to base interpolation manager.
- [base_interp_mgr](#)< rvec_t > * [bim2](#)
Pointer to base interpolation manager for reversed vectors.

Private Member Functions

- [o2scl_interp](#) (const [o2scl_interp](#)< vec_t, rvec_t > &)
- [o2scl_interp](#)< vec_t, rvec_t > & [operator=](#) (const [o2scl_interp](#)< vec_t, rvec_t > &)

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

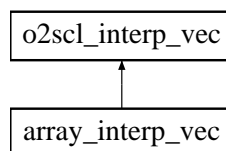
- [interp.h](#)

8.241 o2scl_interp_vec Class Template Reference

Interpolation class for pre-specified vector.

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

Inheritance diagram for o2scl_interp_vec::



8.241.1 Detailed Description

template<class vec_t = ovector_const_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[].

This class does not double check the vector to ensure that all of the intervals for the abscissa are all increasing or all decreasing.

The type of interpolation to be performed can be specified using the [set_type\(\)](#) function. The default is cubic splines with natural boundary conditions.

Todo

Need to fix constructor to behave properly if init() fails. It should free the memory and set `ln` to zero.

Todo

Specify a [base_interp_mgr](#) object instead of [base_interp](#) objects

Definition at line 1389 of file `interp.h`.

Public Member Functions

- [o2scl_interp_vec](#) ([base_interp_mgr](#)< vec_t > &it, size_t n, const vec_t &x, const vec_t &y)
Create with base interpolation object it.
- 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.

Protected Attributes

- [alloc_t](#) [ao](#)
Memory allocator for objects of type `alloc_vec_t`.
- [base_interp](#)< vec_t > * [itp](#)
Base interpolation object.
- [base_interp_mgr](#)< vec_t > * [bim](#)
The interpolation manager.
- 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 y 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.

Private Member Functions

- `o2scl_interp_vec` (const `o2scl_interp_vec`< `vec_t`, `alloc_vec_t`, `alloc_t` > &)
- `o2scl_interp_vec`< `vec_t`, `alloc_vec_t`, `alloc_t` > & **operator=** (const `o2scl_interp_vec`< `vec_t`, `alloc_vec_t`, `alloc_t` > &)

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

- `interp.h`

8.242 ode_bv_multishoot Class Template Reference

Multishooting.

```
#include <ode_bv_multishoot.h>
```

8.242.1 Detailed Description

`template<class param_t, class func_t = ode_func_t<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class vec_int_t = ovector_int_base, class mat_t = omatrix> class ode_bv_multishoot< param_t, func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t, mat_t >`

Multishooting.

Experimental.

Todo

Improve documentation a little and create testing code

Definition at line 48 of file `ode_bv_multishoot.h`.

Public Member Functions

- virtual int **solve** (`vec_t` &mesh, int &n_func, `vec_t` &y_start, `param_t` ¶m, `func_t` &left_b, `func_t` &right_b, `func_t` &extra_b, `func_t` &derivs, `vec_t` &x_save, `mat_t` &y_save)
- int **set_iv** (`ode_iv_solve`< `param_t`, `func_t`, `vec_t`, `alloc_vec_t`, `alloc_t` > &ois)
- int **set_mroot** (`mroot`< `param_t`, `mm_func_t`< `param_t` > > &root)

Data Fields

- `ode_iv_solve`< `param_t`, `func_t`, `vec_t`, `alloc_vec_t`, `alloc_t` > **def_ois**
- `gsl_mroot_hybrids`< `param_t`, `mm_func_t`< `param_t` > > **def_mroot**

Protected Member Functions

- int **solve_fun** (size_t nv, const `vec_t` &sx, `vec_t` &sy, `param_t` &pa)
Function to solve.

Protected Attributes

- `ode_iv_solve`< param_t, func_t, vec_t, alloc_vec_t, alloc_t > * `ois`
The initial value solver.
- `gsl_mroot_hybrids`< param_t, `mm_func_t`< param_t > > * `mrootp`
The equation solver.
- `vec_t` * `l_mesh`
Desc.
- `vec_t` * `l_y_start`
Desc.
- `param_t` * `l_param`
Desc.
- `func_t` * `l_left_b`
Desc.
- `func_t` * `l_right_b`
Desc.
- `func_t` * `l_extra_b`
Desc.
- `func_t` * `l_derivs`
Desc.
- `int` * `l_n_func`
Desc.
- `vec_t` * `l_x_save`
Desc.
- `mat_t` * `l_y_save`
Desc.
- `bool` `save`
Desc.

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

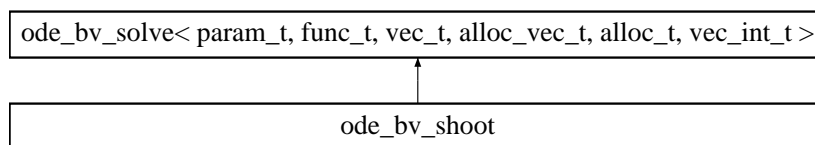
- `ode_bv_multishoot.h`

8.243 ode_bv_shoot Class Template Reference

Solve boundary-value ODE problems by shooting.

```
#include <ode_bv_solve.h>
```

Inheritance diagram for `ode_bv_shoot`:



8.243.1 Detailed Description

```
template<class param_t, class func_t = ode_func_t<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc, class vec_int_t = ovector_int_base> 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.

Idea for future

Create a solution [table](#) as in [ode_iv_solve](#)

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

Public Member Functions

- virtual int [solve](#) (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t ¥d, 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_funct](#)< param_t > > [def_mroot](#)
The default equation solver.

Protected Member Functions

- int [solve_fun](#) (size_t nv, const vec_t &sx, vec_t &sy, param_t &pa)
The shooting function to be solved by the multidimensional solver.

Protected Attributes

- [ode_iv_solve](#)< param_t, func_t, vec_t, alloc_vec_t, alloc_t > * [oisp](#)
The solver for the initial value problem.
- [mroot](#)< param_t, [mm_funct](#)< 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:

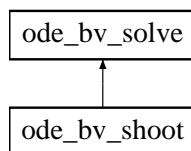
- `ode_bv_solve.h`

8.244 ode_bv_solve Class Template Reference

Solve boundary-value ODE problems with constant boundary conditions [abstract base].

```
#include <ode_bv_solve.h>
```

Inheritance diagram for ode_bv_solve::



8.244.1 Detailed Description

```
template<class param_t, class func_t = ode_func<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc, class vec_int_t = ovector_int_base> class ode_bv_solve< param_t, func_t, vec_t, alloc_vec_t, alloc_t,
vec_int_t >
```

Solve boundary-value ODE problems with constant boundary conditions [abstract base].

Definition at line 44 of file ode_bv_solve.h.

Public Member Functions

- virtual int [solve](#) (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t ¥d, vec_int_t &index, param_t &pa, func_t &derivs)=0
Solve the boundary-value problem.

Data Fields

- 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.

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

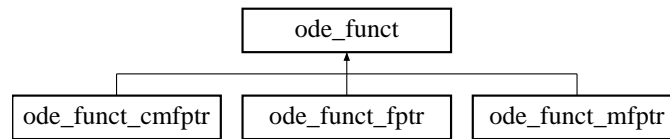
- ode_bv_solve.h

8.245 ode_func Class Template Reference

Ordinary differential equation function [abstract base].

```
#include <ode_func.h>
```

Inheritance diagram for ode_func::



8.245.1 Detailed Description

```
template<class param_t, class vec_t = ovector_base> class ode_func< param_t, vec_t >
```

Ordinary differential equation function [abstract base].

This base class provides the basic format for specifying ordinary differential equations to integrate with the O₂scl ODE solvers. Select the appropriate child of this class according to the kind of functions which are to be given to the solver.

For functions with C-style arrays, use the corresponding children of [ode_vfunc](#).

Definition at line 44 of file ode_func.h.

Public Member Functions

- virtual int [operator\(\)](#) (double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &pa)=0
Compute the nv derivatives as a function of the nv functions specified in y at the point x.

Private Member Functions

- [ode_func](#) (const [ode_func](#) &)
- [ode_func](#) & [operator=](#) (const [ode_func](#) &)

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

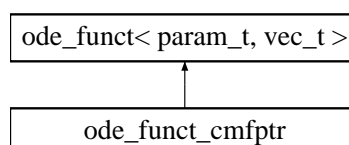
- ode_func.h

8.246 ode_func_cmfprr Class Template Reference

Provide ODE functions in the form of const member function pointers.

```
#include <ode_func.h>
```

Inheritance diagram for ode_func_cmfprr::



8.246.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base> class ode_funct_cmfp< tclass, param_t, vec_t >

Provide ODE functions in the form of const member function pointers.

Definition at line 167 of file ode_funct.h.

Public Member Functions

- [ode_funct_cmfp](#) (tclass *tp, int(tclass::*fp)(double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &) const)
Create an object given a class and member function pointer.
- virtual int [operator\(\)](#) (double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &pa)
Compute the `nv` derivatives as a function of the `nv` functions specified in `y` at the point `x`.

Protected Attributes

- int(tclass::* [fptr](#))(double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &) const
The pointer to the member function.
- tclass * [tptr](#)
The pointer to the class.

Private Member Functions

- [ode_funct_cmfp](#) (const [ode_funct_cmfp](#) &)
- [ode_funct_cmfp](#) & [operator=](#) (const [ode_funct_cmfp](#) &)

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

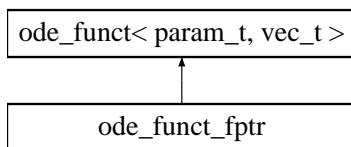
- ode_funct.h

8.247 ode_funct_fptr Class Template Reference

Provide ODE functions in the form of function pointers.

```
#include <ode_funct.h>
```

Inheritance diagram for ode_funct_fptr::



8.247.1 Detailed Description

template<class param_t, class vec_t = ovector_base> class ode_funct_fptr< param_t, vec_t >

Provide ODE functions in the form of function pointers.

Definition at line 73 of file ode_funct.h.

Public Member Functions

- `ode_funct_fptr` (`int(*fp)(double, size_t, const vec_t &, vec_t &, param_t &)`)
Create an object given a function pointer.
- virtual `int operator()` (`double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &pa`)
Compute the `nv` derivatives as a function of the `nv` functions specified in `y` at the point `x`.

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:

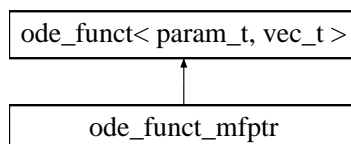
- `ode_funct.h`

8.248 ode_funct_mfptr Class Template Reference

Provide ODE functions in the form of member function pointers.

```
#include <ode_funct.h>
```

Inheritance diagram for `ode_funct_mfptr`:



8.248.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_base> class ode_funct_mfptr< tclass, param_t, vec_t >
```

Provide ODE functions in the form of member function pointers.

Definition at line 118 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 &)`)
Create an object given a class and member function pointer.
- virtual `int operator()` (`double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &pa`)
Compute the `nv` derivatives as a function of the `nv` functions specified in `y` at the point `x`.

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_funct_mfptr` (const `ode_funct_mfptr` &)
- `ode_funct_mfptr` & `operator=` (const `ode_funct_mfptr` &)

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

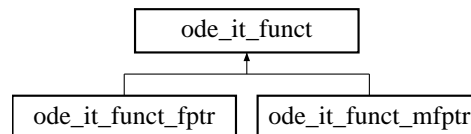
- `ode_funct.h`

8.249 ode_it_funct Class Template Reference

Function class for `ode_it_solve`.

```
#include <ode_it_solve.h>
```

Inheritance diagram for `ode_it_funct`:



8.249.1 Detailed Description

```
template<class vec_t = o2scl::ovector_base> class ode_it_funct< vec_t >
```

Function class for `ode_it_solve`.

Definition at line 45 of file `ode_it_solve.h`.

Public Member Functions

- 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:

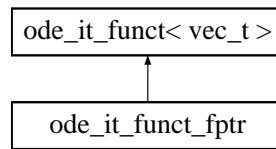
- `ode_it_solve.h`

8.250 ode_it_funct_fptr Class Template Reference

Function pointer for `ode_it_solve`.

```
#include <ode_it_solve.h>
```

Inheritance diagram for ode_it_funct_fptr::



8.250.1 Detailed Description

template<class vec_t = o2scl::ovector_base> class ode_it_funct_fptr< vec_t >

Function pointer for [ode_it_solve](#).

Definition at line 62 of file `ode_it_solve.h`.

Public Member Functions

- [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 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:

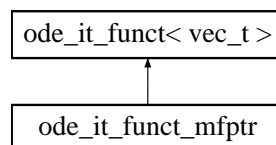
- `ode_it_solve.h`

8.251 ode_it_funct_mfptr Class Template Reference

Member function pointer for [ode_it_solve](#).

`#include <ode_it_solve.h>`

Inheritance diagram for ode_it_funct_mfptr::



8.251.1 Detailed Description

template<class tclass, class vec_t = o2scl::ovector_base> class ode_it_funct_mfptr< tclass, vec_t >

Member function pointer for [ode_it_solve](#).

Definition at line 88 of file `ode_it_solve.h`.

Public Member Functions

- [ode_it_funcn_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 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

8.252 ode_it_solve Class Template Reference

ODE solver using a generic linear solver to solve finite-difference equations.

```
#include <ode_it_solve.h>
```

8.252.1 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

Implement as a child of [ode_bv_solve](#) ?

Todo

Max and average tolerance?

Todo

partial correction option?

Definition at line 125 of file ode_it_solve.h.

Public Member Functions

- int [set_solver](#) (o2scl_linalg::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

- [o2scl_linalg::linear_solver](#)< solver_vec_t, solver_mat_t > * [solver](#)
Solver.

Storage for functions

- [ode_it_funct](#)< vec_t > * [fl](#)
- [ode_it_funct](#)< vec_t > * [fr](#)
- [ode_it_funct](#)< vec_t > * [fd](#)

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

- [ode_it_solve.h](#)

8.253 ode_iv_solve Class Template Reference

Solve an initial-value ODE problems given an adaptive ODE stepper.

```
#include <ode_iv_solve.h>
```

8.253.1 Detailed Description

```
template<class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class
alloc_t = ovector_alloc, class mat_row_t = omatrix_row> class ode_iv_solve< param_t, func_t, vec_t, alloc_vec_t, alloc_t,
mat_row_t >
```

Solve an initial-value ODE problems given an adaptive ODE stepper.

Idea for future

Add error information

Definition at line 44 of file [ode_iv_solve.h](#).

Public Member Functions

- int [set_adapt_step](#) ([adapt_step](#)< param_t, func_t, vec_t, alloc_vec_t, alloc_t > &as)
Set the adaptive stepper to use.
- 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.
- template<class mat_t >
int [solve_grid_derivs](#) (double x0, double x1, double h, size_t n, vec_t &ystart, size_t &nsol, vec_t &xsol, mat_t &ysol, mat_t &dydx_sol, param_t &pa, func_t &derivs)
Solve the initial-value problem from x0 to x1 over a grid storing derivatives.
- int [solve_final_value](#) (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t ¥d, param_t &pa, func_t &derivs)
Solve the initial-value problem to get the final value.
- int [solve_final_value_derivs](#) (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t ¥d, vec_t &dydx_start, vec_t &dydx_end, param_t &pa, func_t &derivs)
Solve the initial-value problem to get the final value and derivative.
- virtual const char * [type](#) ()
Return the type, "ode_iv_solve".

Data Fields

- int [verbose](#)
Set output level.
- size_t [nsteps_out](#)
Number of output points if [verbose](#) is greater than zero (default 10).
- int [ntrial](#)
Maximum number of steps for [solve_final_value\(\)](#) (default 1000).
- bool [exit_on_fail](#)
If true, stop the solution if the adaptive stepper fails.
- [gsl_astep](#)< param_t, func_t, vec_t, alloc_vec_t, alloc_t > [gsl_astp](#)
The default adaptive stepper.

Protected Member Functions

- virtual int [print_iter](#) (double x, size_t nv, vec_t &y)
Print out iteration information.

Protected Attributes

- alloc_vec_t [dydx](#)
Derivative.
- alloc_t [ao](#)
Memory allocator.
- [adapt_step](#)< param_t, func_t, vec_t, alloc_vec_t, alloc_t > * [astp](#)
The adaptive stepper.

8.253.2 Member Function Documentation

8.253.2.1 `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.

If `verbose` is greater than zero, The solution at less than or approximately equal to `nsteps_out` points will be written to `std::cout`.

If `verbose` is greater than one, a character will be required after each selected point.

The solution fails if more than `ntrial` steps are required.

Definition at line 503 of file `ode_iv_solve.h`.

8.253.2.2 `int solve_final_value_derivs (double x0, double x1, double h, size_t n, vec_t & ystart, vec_t & yend, vec_t & dydx_start, vec_t & dydx_end, param_t & pa, func_t & derivs)` `[inline]`

Solve the initial-value problem to get the final value and derivative.

If `verbose` is greater than zero, The solution at less than or approximately equal to `nsteps_out` points will be written to `std::cout`.

If `verbose` is greater than one, a character will be required after each selected point.

The solution fails if more than `ntrial` steps are required.

This function will also fail if $x1 > x0$ and $h < 0$ or if $x1 < x0$ but $h > 0$.

Todo

Add error information

Todo

At present, `dydx_start` is computed, but it should probably be assumed that the user will provide this.

Idea for future

It looks like the $x0 < x1$ code and the $x1 < x0$ code is the same?

Definition at line 534 of file `ode_iv_solve.h`.

8.253.2.3 `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 `matrix` of size `[nsol][n]`. This function never takes a step larger than the grid size, but will take a step smaller than the grid size in order to ensure accuracy.

If `verbose` is greater than zero, The at each grid point will be written to `std::cout`. If `verbose` is greater than one, a character will be required after each point.

Definition at line 200 of file `ode_iv_solve.h`.

8.253.2.4 `int solve_grid_derivs (double x0, double x1, double h, size_t n, vec_t & ystart, size_t nsol, vec_t & xsol, mat_t & ysol, mat_t & dydx_sol, param_t & pa, func_t & derivs)` `[inline]`

Solve the initial-value problem from $x0$ to $x1$ over a grid storing derivatives.

Initially, `xsol` should be an array of size `nsol`, and `ysol` should be a `matrix` of size `[nsol][n]`. This function never takes a step larger than the grid size.

If `verbose` is greater than zero, The solution at each grid point will be written to `std::cout`. If `verbose` is greater than one, a character will be required after each point.

Todo

Add error information

Definition at line 294 of file ode_iv_solve.h.

8.253.2.5 `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 (e.g. `omatrix`) 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*.

If [verbose](#) is greater than zero, The solution at each internal point will be written to `std::cout`. If [verbose](#) is greater than one, a character will be required after each point.

If the given value of *h* is small enough, the solution may generate more points than the space initially allocated and the full solution will not be generated.

Idea for future

Should we give the option not to call the error handler in case running out of space in the [table](#)?

Idea for future

Consider modifying so that this can handle tables which are too small by removing half the rows and doubling the stepsize. Alternatively, make a function which accepts allocation objects and can re-allocate the vector and matrix space if required? (This latter option might be hard for arrays since they are really of fixed size.) A final alternative is to offer an `ovector`/`omatrix` interface which does the reallocation automatically if necessary.

Idea for future

Some copying is done here and it would be nice if that could be avoided.

Definition at line 125 of file ode_iv_solve.h.

8.253.3 Field Documentation**8.253.3.1** `bool exit_on_fail`

If true, stop the solution if the adaptive stepper fails.

If this is false, then failures in the adaptive stepper are ignored.

Definition at line 684 of file ode_iv_solve.h.

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

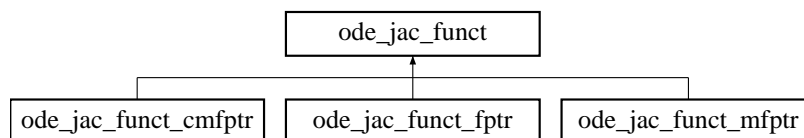
- ode_iv_solve.h

8.254 ode_jac_func Class Template Reference

Ordinary differential equation Jacobian [abstract base].

```
#include <ode_jac_func.h>
```

Inheritance diagram for `ode_jac_func::`



8.254.1 Detailed Description

`template<class param_t, class vec_t = ovector_base, class mat_t = omatrix_base> class ode_jac_funcn< param_t, vec_t, mat_t >`

Ordinary differential equation Jacobian [abstract base].

This base class provides the basic format for specifying the Jacobian for ordinary differential equations to integrate with the O₂scl ODE solvers. Select the appropriate child of this class according to the kind of functions which are to be given to the solver.

For functions with C-style arrays, use the corresponding children of [ode_jac_vfuncn](#).

Definition at line 45 of file ode_jac_funcn.h.

Public Member Functions

- virtual int [operator\(\)](#) (double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx, param_t &pa)=0
Compute the derivatives dfdx and the Jacobian matrix dfdy given y at the point x.

Private Member Functions

- [ode_jac_funcn](#) (const [ode_jac_funcn](#) &)
- [ode_jac_funcn](#) & [operator=](#) (const [ode_jac_funcn](#) &)

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

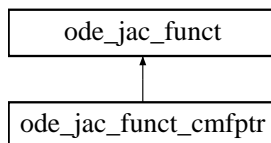
- ode_jac_funcn.h

8.255 ode_jac_funcn_cmfpnr Class Template Reference

Provide ODE Jacobian in the form of const member function pointers.

```
#include <ode_jac_funcn.h>
```

Inheritance diagram for ode_jac_funcn_cmfpnr::



8.255.1 Detailed Description

`template<class tclass, class param_t, class vec_t = ovector_base, class mat_t = omatrix_base> class ode_jac_funcn_cmfpnr< tclass, param_t, vec_t, mat_t >`

Provide ODE Jacobian in the form of const member function pointers.

Definition at line 183 of file ode_jac_funct.h.

Public Member Functions

- `ode_jac_funct_cmfp` (tclass *tp, int(tclass::*fp)(double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx, param_t &) const)
Create an object given a class and member function pointer.
- virtual int `operator()` (double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx, param_t &pa)
Compute the derivatives `dfdx` and the Jacobian matrix `dfdy` given `y` at the point `x`.

Protected Attributes

- int(tclass::* `fptr`)(double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx, param_t &) const
The pointer to the member function.
- tclass * `tptr`
The pointer to the class.

Private Member Functions

- `ode_jac_funct_cmfp` (const `ode_jac_funct_cmfp` &)
- `ode_jac_funct_cmfp` & `operator=` (const `ode_jac_funct_cmfp` &)

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

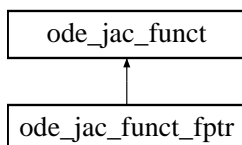
- ode_jac_funct.h

8.256 ode_jac_funct_fptr Class Template Reference

Provide ODE Jacobian in the form of function pointers.

```
#include <ode_jac_funct.h>
```

Inheritance diagram for `ode_jac_funct_fptr`:



8.256.1 Detailed Description

```
template<class param_t, class vec_t = ovector_base, class mat_t = omatrix_base> class ode_jac_funct_fptr< param_t, vec_t, mat_t >
```

Provide ODE Jacobian in the form of function pointers.

Definition at line 75 of file ode_jac_funct.h.

Public Member Functions

- `ode_jac_funct_fptr` (`int(*fp)(double, size_t, const vec_t &, mat_t &, vec_t &, param_t &)`)
Create an object given a function pointer.
- virtual `int operator()` (`double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx, param_t &pa`)
Compute the derivatives $dfdx$ and the Jacobian matrix $dfdy$ given y at the point x .

Protected Attributes

- `int(* fptr)` (`double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx, param_t &`)
The function pointer.

Private Member Functions

- `ode_jac_funct_fptr` (`const ode_jac_funct_fptr &`)
- `ode_jac_funct_fptr & operator=` (`const ode_jac_funct_fptr &`)

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

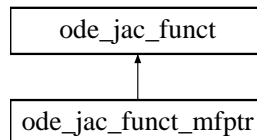
- `ode_jac_funct.h`

8.257 ode_jac_funct_mfptr Class Template Reference

Provide ODE Jacobian in the form of member function pointers.

```
#include <ode_jac_funct.h>
```

Inheritance diagram for `ode_jac_funct_mfptr`:



8.257.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_base, class mat_t = omatrix_base> class ode_jac_funct_mfptr<
tclass, param_t, vec_t, mat_t >
```

Provide ODE Jacobian in the form of member function pointers.

Definition at line 127 of file `ode_jac_funct.h`.

Public Member Functions

- `ode_jac_funct_mfptr` (`tclass *tp, int(tclass::*fp)(double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx, param_t &)`)
Create an object given a class and member function pointer.
- virtual `int operator()` (`double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx, param_t &pa`)
Compute the derivatives $dfdx$ and the Jacobian matrix $dfdy$ given y at the point x .

Protected Attributes

- `int(tclass::* fptr)(double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx, param_t &)`
The pointer to the member function.
- `tclass * tptr`
The pointer to the class.

Private Member Functions

- `ode_jac_funct_mfptr` (const `ode_jac_funct_mfptr` &)
- `ode_jac_funct_mfptr` & `operator=` (const `ode_jac_funct_mfptr` &)

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

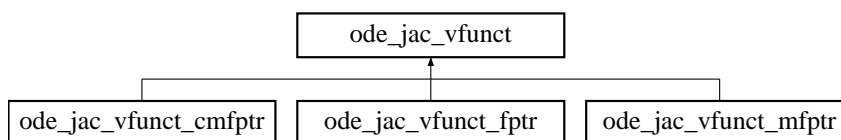
- `ode_jac_funct.h`

8.258 ode_jac_vfunct Class Template Reference

Ordinary differential equation Jacobian with arrays [abstract base].

```
#include <ode_jac_funct.h>
```

Inheritance diagram for `ode_jac_vfunct`:



8.258.1 Detailed Description

```
template<class param_t, size_t nv> class ode_jac_vfunct< param_t, nv >
```

Ordinary differential equation Jacobian with arrays [abstract base].

This base class provides the basic format for specifying ordinary differential equations to integrate with the O2scl ODE solvers. Select the appropriate child of this class according to the kind of functions which are to be given to the solver.

Definition at line 245 of file `ode_jac_funct.h`.

Public Member Functions

- virtual `int operator()` (double x, size_t nvar, const double y[nv], double dfdy[nv][nv], double dfdx[nv], param_t &pa)=0
Compute the derivatives dfdx and the Jacobian matrix dfdy given y at the point x.

Private Member Functions

- `ode_jac_vfunct` (const `ode_jac_vfunct` &)
- `ode_jac_vfunct` & `operator=` (const `ode_jac_vfunct` &)

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

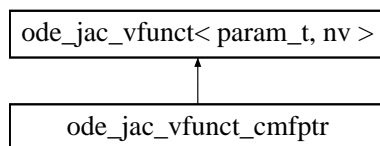
- `ode_jac_funct.h`

8.259 ode_jac_vfunct_cmfpnr Class Template Reference

Provide ODE Jacobian in the form of const member function pointers.

```
#include <ode_jac_funct.h>
```

Inheritance diagram for ode_jac_vfunct_cmfpnr::



8.259.1 Detailed Description

```
template<class tclass, class param_t, size_t nv> class ode_jac_vfunct_cmfpnr< tclass, param_t, nv >
```

Provide ODE Jacobian in the form of const member function pointers.

Definition at line 370 of file ode_jac_funct.h.

Public Member Functions

- [ode_jac_vfunct_cmfpnr](#) (tclass *tp, int(tclass::*fp)(double x, size_t nvar, const double y[nv], double dfdy[nv][nv], double dfdx[nv], param_t &) const)
Specify the member function pointer.
- virtual int [operator\(\)](#) (double x, size_t nvar, const double y[nv], double dfdy[nv][nv], double dfdx[nv], param_t &pa)
Compute the derivatives dfdx and the Jacobian matrix dfdy given y at the point x.

Protected Attributes

- int(tclass::* [fptr](#)) (double x, size_t nvar, const double y[nv], double dfdy[nv][nv], double dfdx[nv], param_t &) const
Pointer to the member function.
- tclass * [tptr](#)
Pointer to the class.

Private Member Functions

- [ode_jac_vfunct_cmfpnr](#) (const [ode_jac_vfunct_cmfpnr](#) &)
- [ode_jac_vfunct_cmfpnr](#) & [operator=](#) (const [ode_jac_vfunct_cmfpnr](#) &)

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

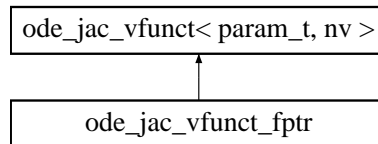
- ode_jac_funct.h

8.260 ode_jac_vfunct_fptr Class Template Reference

Function pointer to a function.

```
#include <ode_jac_funct.h>
```

Inheritance diagram for ode_jac_vfunct_fptr::



8.260.1 Detailed Description

template<class param_t, size_t nv> class ode_jac_vfunct_fptr< param_t, nv >

Function pointer to a function.

Definition at line 274 of file ode_jac_func.h.

Public Member Functions

- [ode_jac_vfunct_fptr](#) (int(*fp)(double, size_t, const double y[nv], double dfdy[nv][nv], double dfdx[nv], param_t &))
Specify the function pointer.
- virtual int [operator\(\)](#) (double x, size_t nvar, const double y[nv], double dfdy[nv][nv], double dfdx[nv], param_t &pa)
Compute the derivatives dfdx and the Jacobian matrix dfdy given y at the point x.

Protected Attributes

- int(* [fptr](#)) (double x, size_t nvar, const double y[nv], double dfdy[nv][nv], double dfdx[nv], param_t &)
The function pointer.

Private Member Functions

- [ode_jac_vfunct_fptr](#) (const [ode_jac_vfunct_fptr](#) &)
- [ode_jac_vfunct_fptr](#) & [operator=](#) (const [ode_jac_vfunct_fptr](#) &)

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

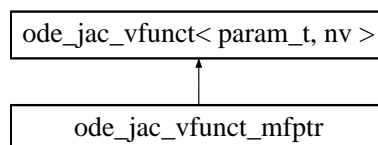
- ode_jac_func.h

8.261 ode_jac_vfunct_mfptr Class Template Reference

Provide ODE Jacobian in the form of member function pointers.

```
#include <ode_jac_func.h>
```

Inheritance diagram for ode_jac_vfunct_mfptr::



8.261.1 Detailed Description

template<class tclass, class param_t, size_t nv> class ode_jac_vfunct_mfptr< tclass, param_t, nv >

Provide ODE Jacobian in the form of member function pointers.

Definition at line 321 of file ode_jac_funct.h.

Public Member Functions

- [ode_jac_vfunct_mfptr](#) (tclass *tp, int(tclass::*fp)(double x, size_t nvar, const double y[nv], double dfdy[nv][nv], double dfdx[nv], param_t &))
Specify the member function pointer.
- virtual int [operator\(\)](#) (double x, size_t nvar, const double y[nv], double dfdy[nv][nv], double dfdx[nv], param_t &pa)
Compute the derivatives dfdx and the Jacobian matrix dfdy given y at the point x.

Protected Attributes

- int(tclass::* [fptr](#))(double x, size_t nvar, const double y[nv], double dfdy[nv][nv], double dfdx[nv], param_t &)
Pointer to the member function.
- tclass * [tptr](#)
Pointer to the class.

Private Member Functions

- [ode_jac_vfunct_mfptr](#) (const [ode_jac_vfunct_mfptr](#) &)
- [ode_jac_vfunct_mfptr](#) & [operator=](#) (const [ode_jac_vfunct_mfptr](#) &)

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

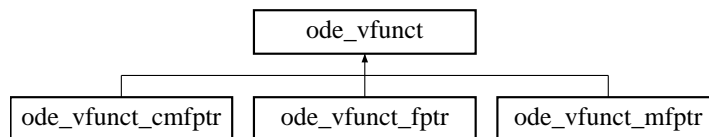
- ode_jac_funct.h

8.262 ode_vfunct Class Template Reference

Ordinary differential equation function with arrays [abstract base].

#include <ode_funct.h>

Inheritance diagram for ode_vfunct::



8.262.1 Detailed Description

template<class param_t, size_t nv> class ode_vfunct< param_t, nv >

Ordinary differential equation function with arrays [abstract base].

This base class provides the basic format for specifying ordinary differential equations to integrate with the O2scl ODE solvers. Select the appropriate child of this class according to the kind of functions which are to be given to the solver.

Definition at line 222 of file ode_funct.h.

Public Member Functions

- virtual int [operator\(\)](#) (double x, size_t nvar, const double y[nv], double dydx[nv], param_t &pa)=0
Compute the nv derivatives as a function of the nv functions specified in y at the point x.

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:

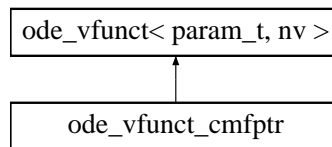
- ode_funct.h

8.263 ode_vfunct_cmfprr Class Template Reference

Provide ODE functions in the form of const member function pointers.

```
#include <ode_funct.h>
```

Inheritance diagram for ode_vfunct_cmfprr::



8.263.1 Detailed Description

```
template<class tclass, class param_t, size_t nv> class ode_vfunct_cmfprr< tclass, param_t, nv >
```

Provide ODE functions in the form of const member function pointers.

Definition at line 345 of file ode_funct.h.

Public Member Functions

- [ode_vfunct_cmfprr](#) (tclass *tp, int(tclass::*fp)(double x, size_t nvar, const double y[nv], double dydx[nv], param_t &) const)
Specify the member function pointer.
- virtual int [operator\(\)](#) (double x, size_t nvar, const double y[nv], double dydx[nv], param_t &pa)
Compute the nv derivatives as a function of the nv functions specified in y at the point x.

Protected Attributes

- int(tclass::* [fptr](#))(double x, size_t nvar, const double y[nv], double dydx[nv], param_t &) const
Pointer to the member function.
- tclass * [tptr](#)
Pointer to the class.

Private Member Functions

- `ode_vfunct_cmfp` (const `ode_vfunct_cmfp` &)
- `ode_vfunct_cmfp` & `operator=` (const `ode_vfunct_cmfp` &)

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

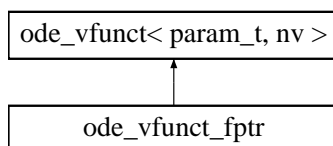
- `ode_funct.h`

8.264 ode_vfunct_fptr Class Template Reference

Function pointer to a function.

```
#include <ode_funct.h>
```

Inheritance diagram for `ode_vfunct_fptr`:



8.264.1 Detailed Description

```
template<class param_t, size_t nv> class ode_vfunct_fptr< param_t, nv >
```

Function pointer to a function.

Definition at line 250 of file `ode_funct.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 int `operator()` (double x, size_t nvar, const double y[nv], double dydx[nv], param_t &pa)
Compute the nv derivatives as a function of the nv functions specified in y at the point x.

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:

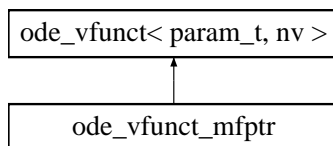
- `ode_funct.h`

8.265 ode_vfunct_mfptr Class Template Reference

Provide ODE functions in the form of member function pointers.

```
#include <ode_funct.h>
```

Inheritance diagram for ode_vfunct_mfptr::



8.265.1 Detailed Description

```
template<class tclass, class param_t, size_t nv> class ode_vfunct_mfptr< tclass, param_t, nv >
```

Provide ODE functions in the form of member function pointers.

Definition at line 296 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 int [operator\(\)](#) (double x, size_t nvar, const double y[nv], double dydx[nv], param_t &pa)
Compute the nv derivatives as a function of the nv functions specified in y at the point x.

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:

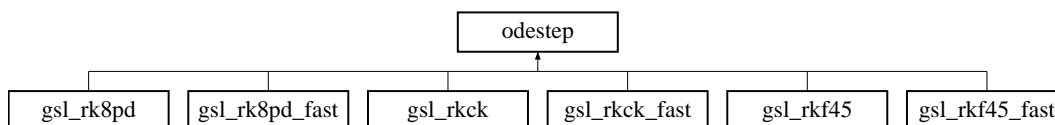
- ode_funct.h

8.266 odestep Class Template Reference

ODE stepper base [abstract base].

```
#include <odestep.h>
```

Inheritance diagram for odestep::



8.266.1 Detailed Description

`template<class param_t, class func_t, class vec_t = ofvector_base> class odestep< param_t, func_t, vec_t >`

ODE stepper base [abstract base].

Note:

This base class does not actually perform any ODE solving use [gsl_rkck](#) or [gsl_rk8pd](#).

Definition at line 40 of file `odestep.h`.

Public Member Functions

- 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)=0
Perform an integration step.

Protected Attributes

- int [order](#)
The order of the ODE stepper.

8.266.2 Member Function Documentation

8.266.2.1 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) [pure 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.

Implemented in [gsl_rk8pd](#), [gsl_rk8pd_fast](#), [gsl_rkck](#), [gsl_rkck_fast](#), [gsl_rkf45](#), and [gsl_rkf45_fast](#).

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

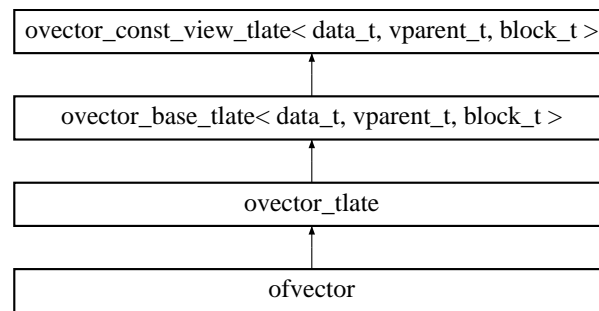
- `odestep.h`

8.267 ofvector Class Template Reference

A vector where the memory allocation is performed in the constructor.

```
#include <ofvector_tlate.h>
```

Inheritance diagram for ofvector::



8.267.1 Detailed Description

template<size_t N = 0> class ofvector< N >

A vector where the memory allocation is performed in the constructor.

This can be useful, for example to easily make C-style arrays of [ovector](#) objects with a fixed size. For example,

```
ofvector<8> x[10];
```

would mean `x` is a 10-dimensional array of [ovector](#) object with initial length 8.

Idea for future

Consider making [allocate\(\)](#) and [free\(\)](#) functions private for this class?

Definition at line 1726 of file `ovector_tlate.h`.

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

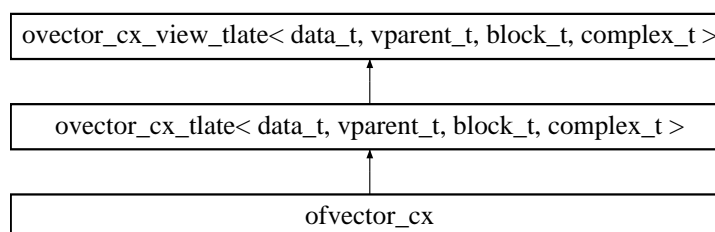
- [ovector_tlate.h](#)

8.268 ofvector_cx Class Template Reference

A complex vector where the memory allocation is performed in the constructor.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ofvector_cx::



8.268.1 Detailed Description

template<size_t N = 0> class ofvector_cx< N >

A complex vector where the memory allocation is performed in the constructor.

Definition at line 1008 of file ovector_cx_tlate.h.

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

- [ovector_cx_tlate.h](#)

8.269 omatrix_alloc Class Reference

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

```
#include <omatrix_tlate.h>
```

8.269.1 Detailed Description

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

Definition at line 1302 of file omatrix_tlate.h.

Public Member Functions

- void [allocate](#) ([omatrix](#) &o, size_t i, size_t j)
Allocate v for i elements.
- void [free](#) ([omatrix](#) &o, size_t i)
Free memory.

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

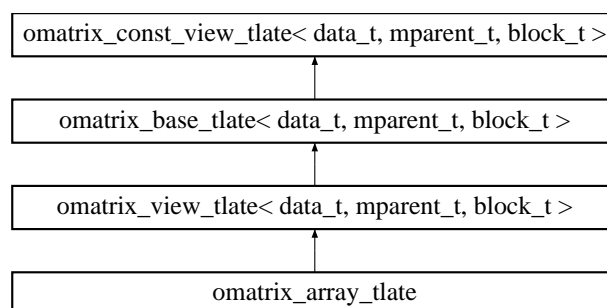
- [omatrix_tlate.h](#)

8.270 omatrix_array_tlate Class Template Reference

Create a matrix from an array.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_array_tlate`:



8.270.1 Detailed Description

template<class data_t, class mparent_t, class block_t> class omatrix_array_tlate< data_t, mparent_t, block_t >

Create a matrix from an array.

Definition at line 1007 of file omatrix_tlate.h.

Public Member Functions

- [omatrix_array_tlate](#) (size_t tot, data_t *dat, size_t start, size_t ltda, size_t sz1, size_t sz2)
Create a vector from dat with size n.

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

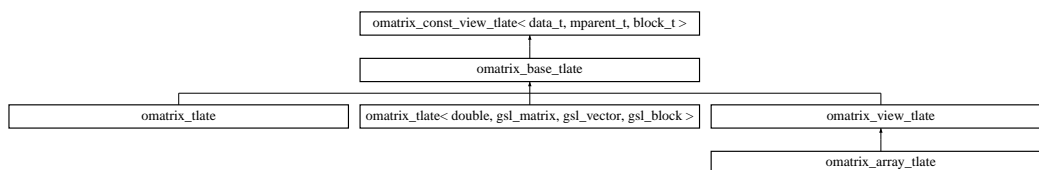
- [omatrix_tlate.h](#)

8.271 omatrix_base_tlate Class Template Reference

A base class for omatrix and omatrix_view.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for omatrix_base_tlate::



8.271.1 Detailed Description

template<class data_t, class mparent_t, class block_t> class omatrix_base_tlate< data_t, mparent_t, block_t >

A base class for omatrix and omatrix_view.

Definition at line 241 of file omatrix_tlate.h.

Public Member Functions

Copy constructors

- [omatrix_base_tlate](#) (const [omatrix_base_tlate](#) &v)
Shallow copy constructor - create a new view of the same matrix.
- [omatrix_base_tlate](#) & [operator=](#) (const [omatrix_base_tlate](#) &v)
Shallow 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_ptr (size_t i, size_t j)`
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`.

Other methods

- `mparent_t * get_gsl_matrix ()`
Return a `gsl` matrix.

Arithmetic

- `omatrix_base_tlate< data_t, mparent_t, block_t > & operator+= (const omatrix_base_tlate< data_t, mparent_t, block_t > &x)`
operator+=
- `omatrix_base_tlate< data_t, mparent_t, block_t > & operator-= (const omatrix_base_tlate< data_t, mparent_t, block_t > &x)`
operator-=
- `omatrix_base_tlate< data_t, mparent_t, block_t > & operator+= (const data_t &y)`
operator+=
- `omatrix_base_tlate< data_t, mparent_t, block_t > & operator-= (const data_t &y)`
operator-=
- `omatrix_base_tlate< data_t, mparent_t, block_t > & operator*= (const data_t &y)`
operator=*

Protected Member Functions

- `omatrix_base_tlate ()`
Desc.

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

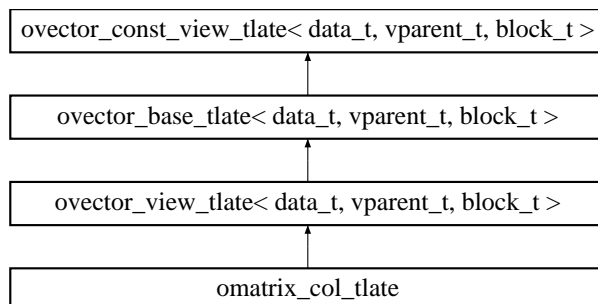
- [omatrix_tlate.h](#)

8.272 **omatrix_col_tlate** Class Template Reference

Create a vector from a column of a matrix.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_col_tlate`:



8.272.1 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 1094 of file `omatrix_tlate.h`.

Public Member Functions

- `omatrix_col_tlate` (`omatrix_base_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:

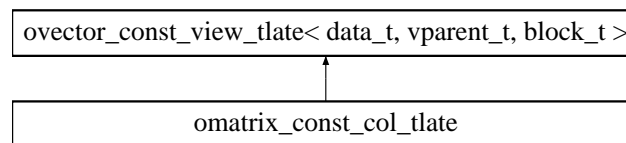
- [omatrix_tlate.h](#)

8.273 `omatrix_const_col_tlate` Class Template Reference

Create a const vector from a column of a matrix.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_const_col_tlate`:



8.273.1 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 1121 of file `omatrix_tlate.h`.

Public Member Functions

- `omatrix_const_col_tlate` (const `omatrix_base_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:

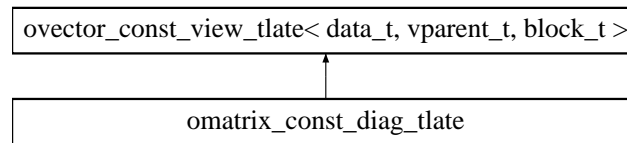
- [omatrix_tlate.h](#)

8.274 `omatrix_const_diag_tlate` Class Template Reference

Create a vector from the main diagonal.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_const_diag_tlate`:



8.274.1 Detailed Description

```
template<class data_t, class mparent_t, class vparent_t, class block_t> class omatrix_const_diag_tlate< data_t, mparent_t, vparent_t, block_t >
```

Create a vector from the main diagonal.

Definition at line 1166 of file `omatrix_tlate.h`.

Public Member Functions

- [`omatrix_const_diag_tlate`](#) (const [`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:

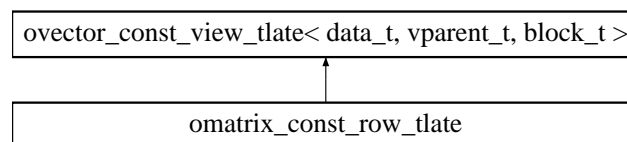
- [omatrix_tlate.h](#)

8.275 `omatrix_const_row_tlate` Class Template Reference

Create a const vector from a row of a matrix.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_const_row_tlate`:



8.275.1 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 1066 of file `omatrix_tlate.h`.

Public Member Functions

- [omatrix_const_row_tlate](#) (const [omatrix_base_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:

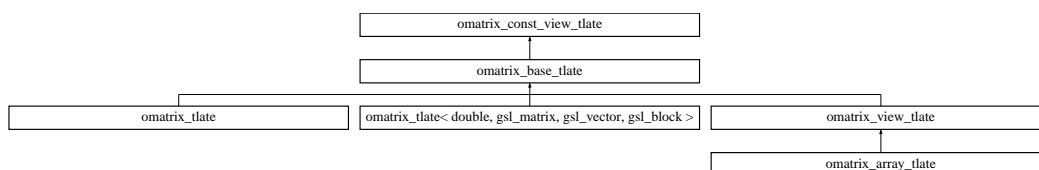
- [omatrix_tlate.h](#)

8.276 `omatrix_const_view_tlate` Class Template Reference

A const matrix view of omatrix objects.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_const_view_tlate`:



8.276.1 Detailed Description

```
template<class data_t, class mparent_t, class block_t> class omatrix_const_view_tlate< data_t, mparent_t, block_t >
```

A const matrix view of omatrix objects.

Definition at line 56 of file `omatrix_tlate.h`.

Public Member Functions

Copy constructors

- [omatrix_const_view_tlate](#) (const [omatrix_const_view_tlate](#) &v)
Shallow copy constructor - create a new view of the same matrix.
- [omatrix_const_view_tlate](#) & [operator=](#) (const [omatrix_const_view_tlate](#) &v)
Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- const data_t * [operator\[\]](#) (size_t i) const
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).
- 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.
- `const mparent_t * get_gsl_matrix_const () const`
Return a const gsl matrix.

Protected Member Functions

- `omatrix_const_view_tlate ()`
Desc.

8.276.2 Member Function Documentation

8.276.2.1 `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 193 of file `omatrix_tlate.h`.

8.276.2.2 `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 "matrix_view", or a "matrix". If `is_owner()` is true, then it is an `omatrix_tlate` object.

Definition at line 216 of file `omatrix_tlate.h`.

8.276.2.3 `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 183 of file `omatrix_tlate.h`.

8.276.2.4 `size_t tda () const` [inline]

Method to return matrix tda.

If no memory has been allocated, this will quietly return zero.

Definition at line 203 of file `omatrix_tlate.h`.

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

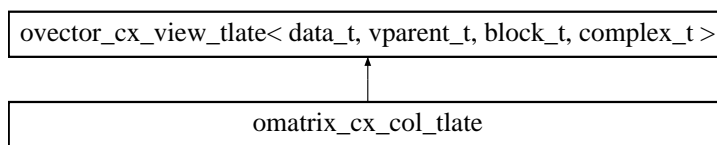
- [omatrix_tlate.h](#)

8.277 **omatrix_cx_col_tlate Class Template Reference**

Create a vector from a column of a matrix.

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for `omatrix_cx_col_tlate`::



8.277.1 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 670 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:

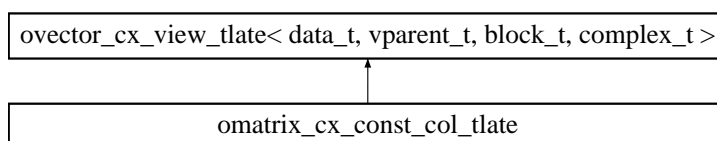
- [omatrix_cx_tlate.h](#)

8.278 `omatrix_cx_const_col_tlate` Class Template Reference

Create a vector from a column of a matrix.

`#include <omatrix_cx_tlate.h>`

Inheritance diagram for `omatrix_cx_const_col_tlate`::



8.278.1 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 690 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:

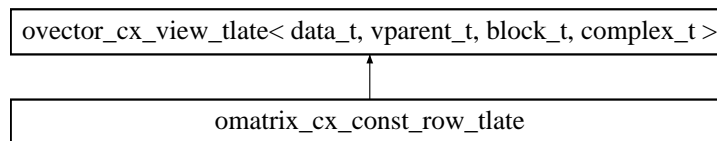
- [omatrix_cx_tlate.h](#)

8.279 `omatrix_cx_const_row_tlate` Class Template Reference

Create a vector from a row of a matrix.

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for `omatrix_cx_const_row_tlate`:



8.279.1 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 650 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:

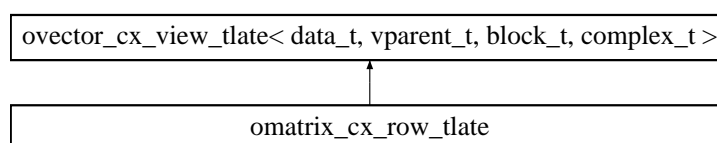
- [omatrix_cx_tlate.h](#)

8.280 `omatrix_cx_row_tlate` Class Template Reference

Create a vector from a row of a matrix.

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for `omatrix_cx_row_tlate`:



8.280.1 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 630 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:

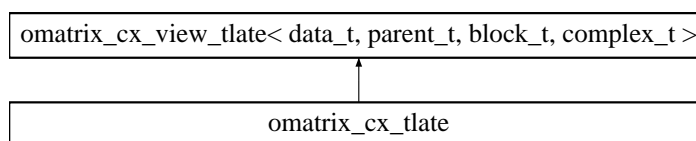
- [omatrix_cx_tlate.h](#)

8.281 `omatrix_cx_tlate` Class Template Reference

A matrix of double-precision numbers.

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for `omatrix_cx_tlate`:



8.281.1 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 398 of file `omatrix_cx_tlate.h`.

Public Member Functions

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 ()`
Compute the transpose.
- `omatrix_cx_tlate< data_t, parent_t, block_t, complex_t > htranspose ()`
Compute the conjugate transpose.

8.281.2 Member Function Documentation

8.281.2.1 `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 584 of file `omatrix_cx_tlate.h`.

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

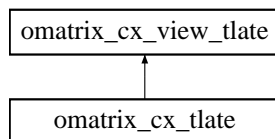
- [omatrix_cx_tlate.h](#)

8.282 `omatrix_cx_view_tlate` Class Template Reference

A matrix view of double-precision numbers.

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for `omatrix_cx_view_tlate`:



8.282.1 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

Copy constructors

- `omatrix_cx_view_tlate (const omatrix_cx_view_tlate &v)`
Shallow copy constructor - create a new view of the same matrix.

- `omatrix_cx_view_tlate & operator= (const omatrix_cx_view_tlate &v)`
Shallow 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.

8.282.2 Member Function Documentation

8.282.2.1 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 356 of file `omatrix_cx_tlate.h`.

8.282.2.2 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 346 of file `omatrix_cx_tlate.h`.

8.282.2.3 `size_t tda () const` `[inline]`

Method to return matrix tda.

If no memory has been allocated, this will quietly return zero.

Definition at line 366 of file `omatrix_cx_tlate.h`.

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

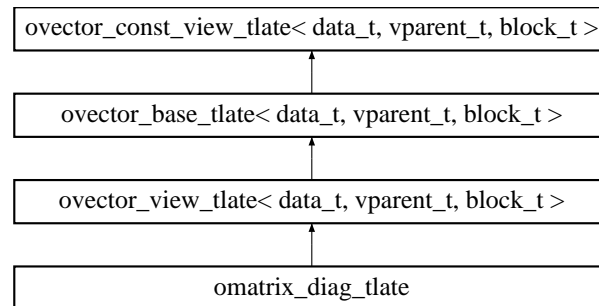
- [omatrix_cx_tlate.h](#)

8.283 **omatrix_diag_tlate** Class Template Reference

Create a vector from the main diagonal.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_diag_tlate`::

**8.283.1** 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 1148 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:

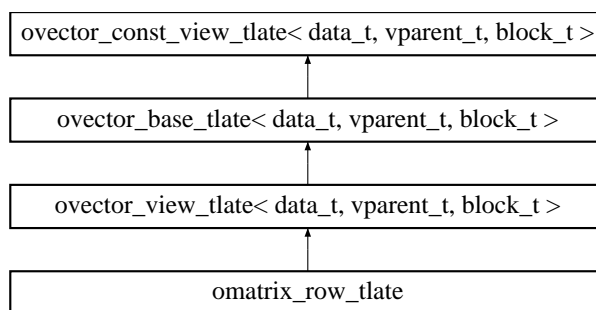
- [omatrix_tlate.h](#)

8.284 **omatrix_row_tlate** Class Template Reference

Create a vector from a row of a matrix.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_row_tlate`::



8.284.1 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 1038 of file omatrix_tlate.h.

Public Member Functions

- [omatrix_row_tlate](#) ([omatrix_base_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:

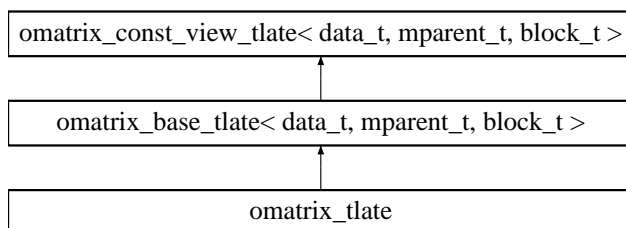
- [omatrix_tlate.h](#)

8.285 omatrix_tlate Class Template Reference

A matrix of double-precision numbers.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for omatrix_tlate::



8.285.1 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.

The basic matrix classes are built upon this template. A matrix of double-precision numbers is an object of type [omatrix](#), which is just a `typedef` defined using this class template. See [Arrays, Vectors, Matrices and Tensors](#) in the User's Guide for more information.

Definition at line 654 of file `omatrix_tlate.h`.

Public Member Functions

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_const_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_const_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_base_tlate](#)< data_t, vparent_t, block_t > ova[])
Deep copy from an array of ovector.
- [omatrix_tlate](#) (size_t n, [uvector_base_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](#) ()
Compute the transpose (even if matrix is not square).

8.285.2 Member Function Documentation

8.285.2.1 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 966 of file `omatrix_tlate.h`.

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

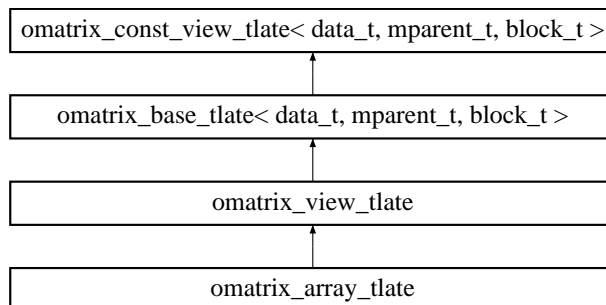
- [omatrix_tlate.h](#)

8.286 `omatrix_view_tlate` Class Template Reference

A matrix view of double-precision numbers.

```
#include <omatrix_tlate.h>
```

Inheritance diagram for `omatrix_view_tlate`::



8.286.1 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.

This is a matrix view, which views a matrix stored somewhere else. For a full matrix template class with its own memory, see [omatrix_tlate](#) . The basic matrix classes are built upon these templates. A matrix of double-precision numbers is an object of type [omatrix](#) . See [Arrays, Vectors, Matrices and Tensors](#) in the User's Guide for more information.

Definition at line 492 of file `omatrix_tlate.h`.

Public Member Functions

Copy constructors

- [omatrix_view_tlate](#) (const [omatrix_view_tlate](#) &v)
Shallow copy constructor - create a new view of the same matrix.
- [omatrix_view_tlate](#) & [operator=](#) (const [omatrix_view_tlate](#) &v)
Shallow copy constructor - create a new view of the same matrix.
- [omatrix_view_tlate](#) ([omatrix_base_tlate](#)< data_t, mparent_t, block_t > &v)
Shallow copy constructor - create a new view of the same matrix.
- [omatrix_view_tlate](#) & [operator=](#) ([omatrix_base_tlate](#)< data_t, mparent_t, block_t > &v)
Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- data_t * [operator\[\]](#) (size_t i) const
Array-like indexing.
- data_t & [operator\(\)](#) (size_t i, size_t j) const
Array-like indexing.
- data_t * [get_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) const
Set (with optional range-checking).
- int [set_all](#) (double val) const
Set all of the value to be the value val.

Protected Member Functions

- [omatrix_view_tlate](#) ()
Desc.

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

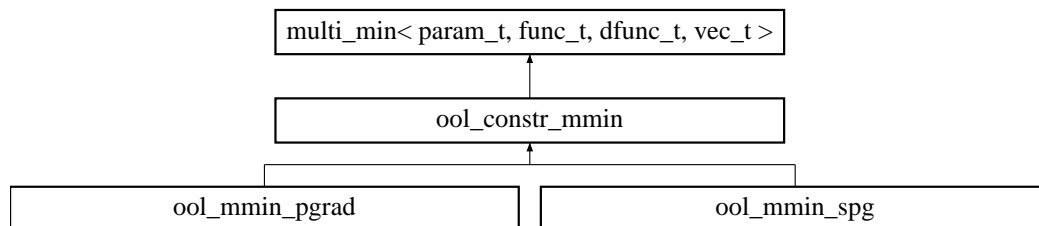
- [omatrix_tlate.h](#)

8.287 ool_constr_mmin Class Template Reference

Constrained multidimensional minimization (OOL) [abstract base].

```
#include <ool_constr_mmin.h>
```

Inheritance diagram for ool_constr_mmin::



8.287.1 Detailed Description

```
template<class param_t, class func_t, class dfunc_t = func_t, class hfunc_t = func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class ool_constr_mmin< param_t, func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t >
```

Constrained multidimensional minimization (OOL) [abstract base].

Todo

Implement automatic computations of [gradient](#) and Hessian

Todo

Construct a non-trivial example for the "examples" directory

Idea for future

Finish [mmin\(\)](#) interface

Definition at line 319 of file ool_constr_mmin.h.

Public Member Functions

- virtual int [allocate](#) (const size_t n)
Allocate memory.
- virtual int [free](#) ()
Free previously allocated memory.
- virtual int [restart](#) ()

Restart the minimizer.

- virtual int [set](#) (func_t &fn, dfunc_t &dfn, vec_t &init, param_t &par)
Set the function, the initial guess, and the parameters.
- virtual int [set_hess](#) (func_t &fn, dfunc_t &dfn, hfunc_t &hfn, vec_t &init, param_t &par)
Set the function, the initial guess, and the parameters.
- virtual int [set_constraints](#) (size_t nc, vec_t &lower, vec_t &upper)
Set the constraints.
- virtual int [iterate](#) ()=0
Perform an iteration.
- virtual int [is_optimal](#) ()=0
See if we're finished.
- virtual int [mmin](#) (size_t nvar, vec_t &xx, double &fmin, param_t &pa, func_t &ff)
Calculate the minimum min of func w.r.t. the array x of size nvar.
- virtual int [mmin_hess](#) (size_t nvar, vec_t &xx, double &fmin, param_t &pa, func_t &ff, dfunc_t &df, hfunc_t &hf)
Calculate the minimum min of ff w.r.t. the array x of size nvar with [gradient](#) df and hessian vector product hf.
- virtual int [mmin_de](#) (size_t nvar, vec_t &xx, double &fmin, param_t &pa, func_t &ff, dfunc_t &df)
Calculate the minimum min of func w.r.t. the array x of size nvar with [gradient](#) dfunc.
- const char * [type](#) ()
Return string denoting type ("ool_constr_mmin").

Static Public Attributes

OOB-specific error codes

- static const int [OOL_UNBOUNDEDF](#) = 1101
Lower unbounded function.
- static const int [OOL_INFEASIBLE](#) = 1102
Infeasible point.
- static const int [OOL_FINNERIT](#) = 1103
Too many inner iterations.
- static const int [OOL_FLSEARCH](#) = 1104
Line search failed.
- static const int [OOL_FDDIR](#) = 1105
Unable to find a descent direction.

Protected Member Functions

- void [shrink](#) (const size_t nind, gsl_vector_uint *Ind, const vec_t &V)
Shrink vector ∇ from the full to the reduced space.
- void [expand](#) (const size_t nind, gsl_vector_uint *Ind, const vec_t &V)
Expand vector ∇ from the reduced to the full space.
- double [calc_f](#) (const size_t nind, gsl_vector_uint *Ind, vec_t &X, vec_t &Xc)
Evaluate the objective function from the reduced space.
- int [calc_g](#) (const size_t nind, gsl_vector_uint *Ind, vec_t &X, vec_t &Xc, vec_t &G)
Compute [gradient](#) in the reduced space.
- int [calc_Hv](#) (const size_t nind, gsl_vector_uint *Ind, vec_t &X, vec_t &Xc, vec_t &V, vec_t &Hv)
Evaluate a hessian times a vector from the reduced space.

Protected Attributes

- double [f](#)
The current function value.
- double [size](#)
Desc.
- alloc_t [ao](#)
Memory allocation object.
- alloc_vec_t [x](#)

The current minimum vector.

- `alloc_vec_t` [gradient](#)

The current [gradient](#) vector.

- `alloc_vec_t` [dx](#)

Desc.

- `size_t` [fcount](#)

Number of function evaluations.

- `size_t` [gcount](#)

Number of [gradient](#) evaluations.

- `size_t` [hcount](#)

Number of Hessian evaluations.

- `size_t` [dim](#)

Number of parameters.

- `size_t` [nconstr](#)

Number of constraints.

- `func_t` * [func](#)

User-supplied function.

- `dfunc_t` * [dfunc](#)

Gradient function.

- `hfunc_t` * [hfunc](#)

Hessian function.

- `param_t` * [param](#)

User-specified parameters.

- `alloc_vec_t` [L](#)

Lower bound constraints.

- `alloc_vec_t` [U](#)

Upper bound constraints.

- `bool` [requires_hess](#)

If true, the algorithm requires the hessian vector product.

8.287.2 Member Function Documentation

8.287.2.1 `int calc_Hv (const size_t nind, gsl_vector_uint * Ind, vec_t & X, vec_t & Xc, vec_t & V, vec_t & Hv)` [`inline`, `protected`]

Evaluate a hessian times a vector from the reduced space.

Expand to full space

Definition at line 474 of file `ool_constr_mmin.h`.

8.287.2.2 `virtual int mmin (size_t nvar, vec_t & xx, double & fmin, param_t & pa, func_t & ff)` [`inline`, `virtual`]

Calculate the minimum `min` of `func` w.r.t. the array `x` of size `nvar`.

Todo

Need to finish this function somehow since it's pure virtual in [multi_min](#).

Implements [multi_min](#).

Definition at line 625 of file `ool_constr_mmin.h`.

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

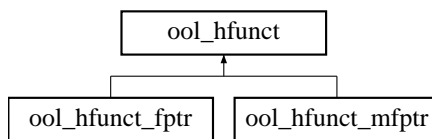
- `ool_constr_mmin.h`

8.288 ool_hfunct Class Template Reference

Hessian product function for [ool_constr_mmin](#) [abstract base].

```
#include <ool_constr_mmin.h>
```

Inheritance diagram for ool_hfunct::



8.288.1 Detailed Description

```
template<class param_t, class vec_t = ovector_base> class ool_hfunct< param_t, vec_t >
```

Hessian product function for [ool_constr_mmin](#) [abstract base].

Definition at line 62 of file ool_constr_mmin.h.

Public Member Functions

- virtual int [operator\(\)](#) (size_t nv, const vec_t &x, const vec_t &v, vec_t &hv, param_t &pa)=0
Evaluate $H(x) \cdot v$.

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

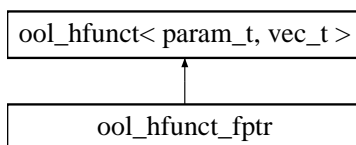
- ool_constr_mmin.h

8.289 ool_hfunct_fptr Class Template Reference

A hessian product supplied by a function pointer.

```
#include <ool_constr_mmin.h>
```

Inheritance diagram for ool_hfunct_fptr::



8.289.1 Detailed Description

```
template<class param_t, class vec_t = ovector_base> class ool_hfunct_fptr< param_t, vec_t >
```

A hessian product supplied by a function pointer.

Definition at line 89 of file ool_constr_mmin.h.

Public Member Functions

- [ool_hfunct_fptr](#) (int(*fp)(size_t nv, const vec_t &x, const vec_t &v, vec_t &hv, param_t &pa))
Specify the function pointer.
- virtual int [operator\(\)](#) (size_t nv, const vec_t &x, const vec_t &v, vec_t &hv, param_t &pa)
Evaluate $H(x) \cdot v$.

Protected Attributes

- int(* [fptr](#))(size_t nv, const vec_t &x, const vec_t &v, vec_t &hv, param_t &pa)
Store the function pointer.

Private Member Functions

- [ool_hfunct_fptr](#) (const [ool_hfunct_fptr](#) &)
- [ool_hfunct_fptr](#) & [operator=](#) (const [ool_hfunct_fptr](#) &)

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

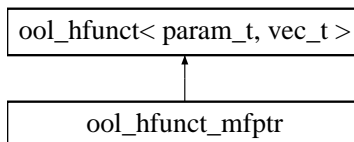
- ool_constr_mmin.h

8.290 ool_hfunct_mfptr Class Template Reference

A hessian product supplied by a member function pointer.

```
#include <ool_constr_mmin.h>
```

Inheritance diagram for ool_hfunct_mfptr::



8.290.1 Detailed Description

```
template<class tclass, class param_t, class vec_t = ovector_base> class ool_hfunct_mfptr< tclass, param_t, vec_t >
```

A hessian product supplied by a member function pointer.

Definition at line 135 of file ool_constr_mmin.h.

Public Member Functions

- [ool_hfunct_mfptr](#) (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, const vec_t &v, vec_t &hv, param_t &pa))
Specify the class instance and member function pointer.
- virtual int [operator\(\)](#) (size_t nv, const vec_t &x, const vec_t &v, vec_t &hv, param_t &pa)
Evaluate $H(x) \cdot v$.

Protected Attributes

- `int(tclass::* fptr)(size_t nv, const vec_t &x, const vec_t &v, vec_t &hv, param_t &pa)`
Store the function pointer.
- `tclass * tptr`
Store a pointer to the class instance.

Private Member Functions

- `ool_hvfunct_mfptr` (const [ool_hvfunct_mfptr](#) &)
- `ool_hvfunct_mfptr & operator=` (const [ool_hvfunct_mfptr](#) &)

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

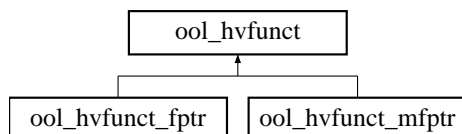
- `ool_constr_mmin.h`

8.291 ool_hvfunct Class Template Reference

Hessian product function base for [ool_constr_mmin](#) using arrays.

```
#include <ool_constr_mmin.h>
```

Inheritance diagram for `ool_hvfunct`:



8.291.1 Detailed Description

```
template<class param_t, size_t nvar> class ool_hvfunct< param_t, nvar >
```

Hessian product function base for [ool_constr_mmin](#) using arrays.

Definition at line 185 of file `ool_constr_mmin.h`.

Public Member Functions

- `virtual int operator\(\) (size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa)=0`
Evaluate $H(x) \cdot v$.

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

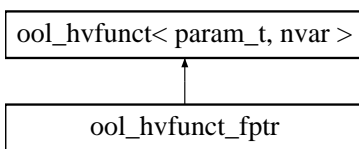
- `ool_constr_mmin.h`

8.292 ool_hvfunct_fptr Class Template Reference

A hessian product supplied by a function pointer using arrays.

```
#include <ool_constr_mmin.h>
```

Inheritance diagram for ool_hvfunct_fptr::



8.292.1 Detailed Description

template<class param_t, size_t nvar> class ool_hvfunct_fptr< param_t, nvar >

A hessian product supplied by a function pointer using arrays.

Definition at line 213 of file ool_constr_mmin.h.

Public Member Functions

- [ool_hvfunct_fptr](#) (int(*fp)(size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa))
Specify the function pointer.
- virtual int [operator\(\)](#) (size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa)
Evaluate $H(x) \cdot v$.

Protected Attributes

- int(* [fptr](#)) (size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa)
Store the function pointer.

Private Member Functions

- [ool_hvfunct_fptr](#) (const [ool_hvfunct_fptr](#) &)
- [ool_hvfunct_fptr](#) & [operator=](#) (const [ool_hvfunct_fptr](#) &)

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

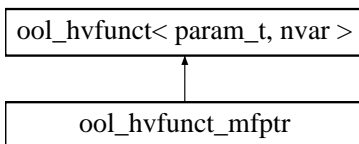
- ool_constr_mmin.h

8.293 ool_hvfunct_mfptr Class Template Reference

A hessian product supplied by a member function pointer using arrays.

#include <ool_constr_mmin.h>

Inheritance diagram for ool_hvfunct_mfptr::



8.293.1 Detailed Description

template<class tclass, class param_t, size_t nvar> class ool_hvfunct_mfptr< tclass, param_t, nvar >

A hessian product supplied by a member function pointer using arrays.

Definition at line 262 of file ool_constr_mmin.h.

Public Member Functions

- **ool_hvfunct_mfptr** (tclass *tp, int(tclass::*fp)(size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa))
Specify the member function pointer.
- virtual int **operator()** (size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa)
Evaluate $H(x) \cdot v$.

Protected Attributes

- int(tclass::* **fptr**) (size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa)
Store the function pointer.
- tclass * **tptr**
Store a pointer to the class instance.

Private Member Functions

- **ool_hvfunct_mfptr** (const **ool_hvfunct_mfptr** &)
- **ool_hvfunct_mfptr** & **operator=** (const **ool_hvfunct_mfptr** &)

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

- ool_constr_mmin.h

8.294 ool_mmin_gencan Class Template Reference

Constrained minimization by the "GENCAN" method (OOL).

```
#include <ool_mmin_gencan.h>
```

8.294.1 Detailed Description

template<class param_t, class func_t, class dfunc_t = func_t, class hfunc_t = func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class ool_mmin_gencan< param_t, func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t >

Constrained minimization by the "GENCAN" method (OOL).

Note:

Not working yet

Definition at line 69 of file ool_mmin_gencan.h.

Public Member Functions

- virtual int [alloc](#) (const size_t n)
Allocate memory.
- virtual int [free](#) ()
Free previously allocated memory.
- virtual int [set](#) (func_t &fn, dfunc_t &dfn, hfunc_t &hfn, vec_t &init, param_t &par)
Set the function, the initial guess, and the parameters.
- virtual int [restart](#) ()
Restart the minimizer.
- virtual int [iterate](#) ()
Perform an iteration.
- virtual int [is_optimal](#) ()
See if we're finished.
- const char * [type](#) ()
Return string denoting type ("ool_mmin_gencan").

Data Fields

- double [epsgpen](#)
Tolerance on Euclidean norm of projected [gradient](#) (default 1.0e-5).
- double [epsgpsn](#)
Tolerance on infinite norm of projected [gradient](#) (default 1.0e-5).
- double [fmin](#)
Minimum function value (default 10^{-99}).
- double [udelta0](#)
Trust-region radius (default -1.0).
- double [ucgmia](#)
Maximum iterations per variable (default -1.0).
- double [ucgmib](#)
Extra maximum iterations (default -1.0).
- int [cg_scre](#)
Conjugate [gradient](#) condition type (default 1).
- double [cg_gpnf](#)
Projected [gradient](#) norm (default 1.0e-5).
- double [cg_epsilon](#)
Desc (default 1.0e-1).
- double [cg_epsf](#)
Desc (default 1.0e-5).
- double [cg_epsnqmp](#)
Stopping fractional tolerance for conjugate [gradient](#) (default 1.0e-4).
- int [cg_maxitnqmp](#)
Maximum iterations for conjugate [gradient](#) (default 5).
- int [nearlyq](#)
Set to 1 if the function is nearly quadratic (default 0).
- double [nint](#)
Interpolation constant (default 2.0).
- double [next](#)
Extrapolation constant (default 2.0).
- int [mininterp](#)
Minimum interpolation size (default 4).
- int [maxextrap](#)
Maximum extrapolations in truncated Newton direction (default 100).
- int [trtype](#)
Type of trust region (default 0).
- double [eta](#)
Threshold for abandoning current face (default 0.9).

- double [delmin](#)
Minimum trust region for truncated Newton direction (default 0.1).
- double [lspgmi](#)
Minimum spectral steplength (default 1.0e-10).
- double [lspgma](#)
Maximum spectral steplength (default 1.0e10).
- double [theta](#)
Constant for the angle condition (default 1.0e-6).
- double [gamma](#)
Constant for Armijo condition (default 1.0e-4).
- double [beta](#)
Constant for beta condition (default 0.5).
- double [sigma1](#)
Lower bound to the step length reduction (default 0.1).
- double [sigma2](#)
Upper bound to the step length reduction (default 0.9).
- double [epsrel](#)
Relative small number (default 1.0e-7).
- double [epsabs](#)
Absolute small number (default 1.0e-10).
- double [infrel](#)
Relative infinite number (default 1.0e20).
- double [infabs](#)
Absolute infinite number (default 1.0e99).

Protected Attributes

- double [cg_src](#)
Desc (default 1.0).
- [alloc_vec_t S](#)
Temporary vector.
- [alloc_vec_t Y](#)
Temporary vector.
- [alloc_vec_t D](#)
Temporary vector.
- [alloc_vec_t cg_W](#)
Temporary vector.
- [alloc_vec_t cg_R](#)
Temporary vector.
- [alloc_vec_t cg_D](#)
Temporary vector.
- [alloc_vec_t cg_Sprev](#)
Temporary vector.
- [alloc_vec_t Xtrial](#)
Temporary vector.
- [alloc_vec_t tnls_Xtemp](#)
Temporary vector.
- [alloc_vec_t near_l](#)
Temporary vector.
- [alloc_vec_t near_u](#)
Temporary vector.
- int * [Ind](#)
Desc.

8.294.2 Field Documentation

8.294.2.1 double fmin

Minimum function value (default 10^{-99}).

If the function value is below this value, then the algorithm assumes that the function is not bounded and exits.

Definition at line 707 of file ool_mmin_gencan.h.

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

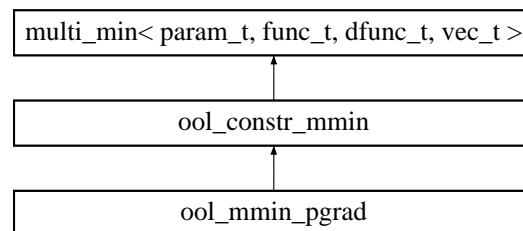
- ool_mmin_gencan.h

8.295 ool_mmin_pgrad Class Template Reference

Constrained minimization by the projected [gradient](#) method (OOL).

```
#include <ool_mmin_pgrad.h>
```

Inheritance diagram for ool_mmin_pgrad::



8.295.1 Detailed Description

```
template<class param_t, class func_t, class dfunc_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t =
ovector_alloc> class ool_mmin_pgrad< param_t, func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >
```

Constrained minimization by the projected [gradient](#) method (OOL).

Todo

Complete the [mmin\(\)](#) interface with automatic [gradient](#)

Todo

Replace the explicit norm computation below with the more accurate dnorm2 from linalg

Definition at line 71 of file ool_mmin_pgrad.h.

Public Member Functions

- virtual int [allocate](#) (const size_t n)
Allocate memory.
- virtual int [free](#) ()
Free previously allocated memory.
- virtual int [set](#) (func_t &fn, dfunc_t &dfn, vec_t &init, param_t &par)
Set the function, the initial guess, and the parameters.
- virtual int [restart](#) ()

Restart the minimizer.

- virtual int [iterate](#) ()
Perform an iteration.
- virtual int [is_optimal](#) ()
See if we're finished.
- const char * [type](#) ()
Return string denoting type ("ool_mmin_pgrad").

Data Fields

- double [fmin](#)
Minimum function value (default 10^{-99}).
- double [tol](#)
Tolerance on infinite norm.
- double [alpha](#)
Constant for the sufficient decrease condition (default 10^{-4}).
- double [sigma1](#)
Lower bound to the step length reduction.
- double [sigma2](#)
Upper bound to the step length reduction.

Protected Types

- typedef [ool_hfunct](#)< int > [hfunc_t](#)
A convenient typedef for the unused Hessian product type.

Protected Member Functions

- int [proj](#) (vec_t &xt)
Project into feasible region.
- int [line_search](#) ()
Line search.

Protected Attributes

- [alloc_vec_t](#) [xx](#)
Temporary vector.

8.295.2 Field Documentation

8.295.2.1 double fmin

Minimum function value (default 10^{-99}).

If the function value is below this value, then the algorithm assumes that the function is not bounded and exits.

Definition at line 168 of file ool_mmin_pgrad.h.

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

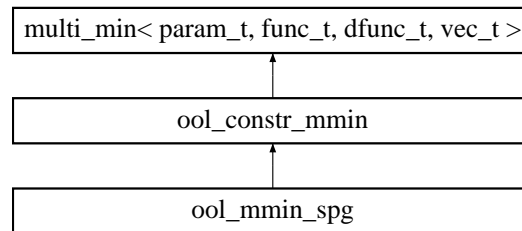
- ool_mmin_pgrad.h

8.296 ool_mmin_spg Class Template Reference

Constrained minimization by the spectral projected [gradient](#) method (OOL).

```
#include <ool_mmin_spg.h>
```

Inheritance diagram for ool_mmin_spg::



8.296.1 Detailed Description

```
template<class param_t, class func_t, class dfunc_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t =
ovector_alloc> class ool_mmin_spg< param_t, func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >
```

Constrained minimization by the spectral projected [gradient](#) method (OOL).

Definition at line 67 of file ool_mmin_spg.h.

Public Member Functions

- virtual int [allocate](#) (const size_t n)
Allocate memory.
- virtual int [free](#) ()
Free previously allocated memory.
- virtual int [set](#) (func_t &fn, dfunc_t &dfn, vec_t &init, param_t &par)
Set the function, the initial guess, and the parameters.
- virtual int [restart](#) ()
Restart the minimizer.
- virtual int [iterate](#) ()
Perform an iteration.
- virtual int [is_optimal](#) ()
See if we're finished.
- const char * [type](#) ()
Return string denoting type ("ool_mmin_spg").

Data Fields

- double [fmin](#)
Minimum function value (default 10^{-99}).
- double [tol](#)
Tolerance on infinite norm (default 10^{-4}).
- double [alphamin](#)
Lower bound to spectral step size (default 10^{-30}).
- double [alphamax](#)
Upper bound to spectral step size (default 10^{30}).
- double [gamma](#)
Sufficient decrease parameter (default 10^{-4}).
- double [sigma1](#)

- *Lower bound to the step length reduction (default 0.1).*
- double [sigma2](#)
Upper bound to the step length reduction (default 0.9).
- size_t [M](#)
Monotonicity parameter (M=1 forces monotonicity) (default 10).

Protected Types

- typedef [ool_hfunc_t](#) < int > [hfunc_t](#)
A convenient typedef for the unused Hessian product type.

Protected Member Functions

- int [line_search](#) ()
Line search.
- int [proj](#) (vec_t &xt)
Project into feasible region.

Protected Attributes

- double [alpha](#)
Armijo parameter.
- alloc_vec_t [xx](#)
Temporary vector.
- alloc_vec_t [d](#)
Temporary vector.
- alloc_vec_t [s](#)
Temporary vector.
- alloc_vec_t [y](#)
Temporary vector.
- alloc_vec_t [fvec](#)
Temporary vector.
- size_t [m](#)
Non-monotone parameter.
- int [tail](#)
Desc.

8.296.2 Field Documentation

8.296.2.1 double fmin

Minimum function value (default 10^{-99}).

If the function value is below this value, then the algorithm assumes that the function is not bounded and exits.

Definition at line 222 of file ool_mmin_spg.h.

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

- ool_mmin_spg.h

8.297 other_todos_and_bugs Class Reference

An empty class to add some items to the todo and bug lists.

```
#include <main.h>
```

8.297.1 Detailed Description

An empty class to add some items to the todo and bug lists.

Todo

- The o2scl-test and o2scl-examples targets require grep, awk, tail, cat, and wc. It would be good to reduce this list to ensure better compatibility.
- More examples and benchmarks
- There are a couple classes which Doxygen doesn't yet parse properly for the class hierarchy: [gsl_root_stef](#), and [ool_mmin_gencan](#). Also should double check [smart_interp](#), etc.
- Make sure abs(-double) isn't allowed by the O2scl and GSL headers with the correct flags

Idea for future

- Fix the PNG images so that they're smaller and repair the transparency issue (probably can be done just using different arguments to 'convert' in the pngfix target)
- Make sure we have a `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

Todo

At present, the testing code assumes that shared libraries are installed. Can this be improved? (12/3/08 - I'm not sure what the status is on this...this should be checked.)

Todo

Make sure default template parameters are documented in each class, and make sure default template parameters exist where they should.

Idea for future

Consider breaking documentation up into sections?

Bug

- 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` (1/4/09 - I'm not sure I remember why this is a problem, but it probably has to do with the detection of libraries which is done in `configure.ac`).

Definition at line 2369 of file `main.h`.

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

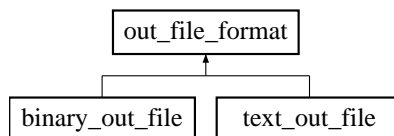
- `main.h`

8.298 out_file_format Class Reference

Class for output file formats [abstract base].

```
#include <file_format.h>
```

Inheritance diagram for out_file_format::



8.298.1 Detailed Description

Class for output file formats [abstract base].

This class is experimental.

Definition at line 49 of file file_format.h.

Public Member Functions

- 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

8.299 ovector_alloc Class Reference

A simple class to provide an `allocate()` function for `ovector`.

```
#include <ovector_tlate.h>
```

8.299.1 Detailed Description

A simple class to provide an `allocate()` function for `ovector`.

Idea for future

Could (or should?) the `allocate()` functions be adapted to return an integer error value?

Definition at line 1690 of file `ovector_tlate.h`.

Public Member Functions

- void `allocate` (`ovector` &o, size_t i)
Allocate \forall for i elements.
- void `free` (`ovector` &o)
Free memory.

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

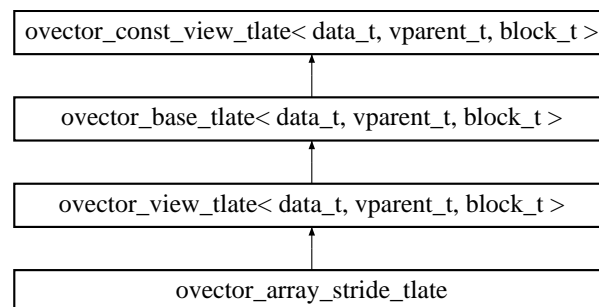
- `ovector_tlate.h`

8.300 ovector_array_stride_tlate Class Template Reference

Create a vector from an array with a stride.

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ovector_array_stride_tlate`:



8.300.1 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 1442 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:

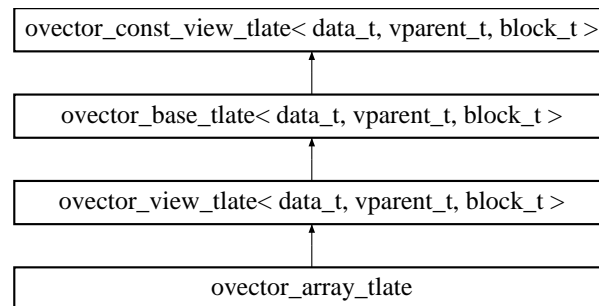
- [ovector_tlate.h](#)

8.301 ovector_array_tlate Class Template Reference

Create a vector from an array.

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_array_tlate::



8.301.1 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 1416 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](#)

8.302 ovector_base_tlate Class Template Reference

A base class for ovector and ovector_view.

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_base_tlate::



8.302.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_base_tlate< data_t, vparent_t, block_t >

A base class for ovector and ovector_view.

This class provides a base class for ovector and ovector_view, mostly useful for creating function arguments which accept either ovector or ovector_view.

Definition at line 472 of file ovector_tlate.h.

Public Member Functions

Copy constructors

- [ovector_base_tlate](#) ([ovector_base_tlate](#) &v)
Shallow copy constructor - create a new view of the same vector.
- [ovector_base_tlate](#) & [operator=](#) ([ovector_base_tlate](#) &v)
Shallow copy constructor - create a new view of the same vector.

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 with operator().
- data_t & [operator\[\]](#) (size_t i)
Array-like indexing.
- data_t & [operator\(\)](#) (size_t i)
Array-like indexing with operator().
- 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 [set_all](#) (double val)
Set all of the value to be the value val.

Arithmetic

- [ovector_base_tlate](#)< data_t, vparent_t, block_t > & [operator+=](#) (const [ovector_base_tlate](#)< data_t, vparent_t, block_t > &x)
operator+=
- [ovector_base_tlate](#)< data_t, vparent_t, block_t > & [operator-=](#) (const [ovector_base_tlate](#)< data_t, vparent_t, block_t > &x)
operator-=
- [ovector_base_tlate](#)< data_t, vparent_t, block_t > & [operator+=](#) (const data_t &x)
operator+=

- `ovector_base_tlate< data_t, vparent_t, block_t > & operator-= (const data_t &x)`
`operator-=`
- `ovector_base_tlate< data_t, vparent_t, block_t > & operator*-= (const data_t &y)`
`operator*-=`

Other methods

- `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_base_tlate ()`
Empty constructor for use by children.

8.302.2 Member Function Documentation

8.302.2.1 `ovector_base_tlate<data_t,vparent_t,block_t>& operator*= (const data_t &y)` [inline]

`operator*=`

If the vector is empty, this function does not perform any modifications and does not call the error handler.

Definition at line 687 of file `ovector_tlate.h`.

8.302.2.2 `ovector_base_tlate<data_t,vparent_t,block_t>& operator+= (const data_t &x)` [inline]

`operator+=`

If the vector is empty, this function does not perform any modifications and does not call the error handler.

Definition at line 664 of file `ovector_tlate.h`.

8.302.2.3 `ovector_base_tlate<data_t,vparent_t,block_t>& operator+= (const ovector_base_tlate< data_t, vparent_t, block_t > &x)` [inline]

`operator+=`

This operator only operates on elements which are present in both vectors, i.e. elements which are present in one vector but missing in the other will be ignored. If one of the two vectors is empty, this function does nothing and does not call the error handler.

Definition at line 632 of file `ovector_tlate.h`.

8.302.2.4 `ovector_base_tlate<data_t,vparent_t,block_t>& operator-= (const data_t &x)` [inline]

`operator-=`

If the vector is empty, this function does not perform any modifications and does not call the error handler.

Definition at line 675 of file `ovector_tlate.h`.

8.302.2.5 `ovector_base_tlate<data_t,vparent_t,block_t>& operator-= (const ovector_base_tlate< data_t, vparent_t, block_t > &x)` [inline]

`operator-=`

This operator only operates on elements which are present in both vectors, i.e. elements which are present in one vector but missing in the other will be ignored. If one of the two vectors is empty, this function does nothing and does not call the error handler.

Definition at line 650 of file `ovector_tlate.h`.

8.302.2.6 `int set_all(double val)` `[inline]`

Set all of the value to be the value `val`.

If the vector is empty, this function does not perform any assignment and does not call the error handler.

Reimplemented in [ovector_reverse_tlate](#), and [ovector_subvector_reverse_tlate](#).

Definition at line 612 of file `ovector_tlate.h`.

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

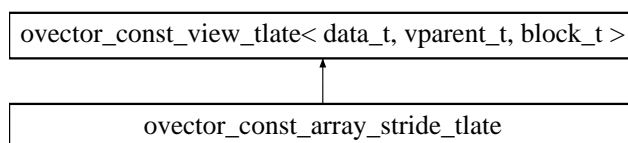
- [ovector_tlate.h](#)

8.303 `ovector_const_array_stride_tlate` Class Template Reference

Create a const vector from an array with a stride.

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ovector_const_array_stride_tlate`:

**8.303.1** 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.

The constructor will fail if the size argument `n` is zero, the stride `s` is zero, or if the pointer `dat` is 0.

Definition at line 1544 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`.

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

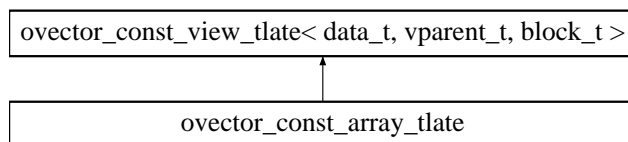
- [ovector_tlate.h](#)

8.304 `ovector_const_array_tlate` Class Template Reference

Create a const vector from an array.

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ovector_const_array_tlate`:



8.304.1 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.

The constructor will fail if the size argument `n` is zero or if the pointer `dat` is 0.

Definition at line 1507 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`.

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

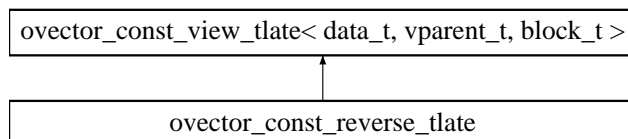
- [ovector_tlate.h](#)

8.305 ovector_const_reverse_tlate Class Template Reference

Reversed view of a vector.

```
#include <ovector_rev_tlate.h>
```

Inheritance diagram for `ovector_const_reverse_tlate::`



8.305.1 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.

Warning:

At present, reversing a reversed vector does not give the original ordering.

Idea for future

I think that maybe in order to ensure that this isn't created from an already reversed vector, this class has to be separated from the hierarchy altogether. However, I think this might break the smart interpolation stuff.

Definition at line 235 of file `ovector_rev_tlate.h`.

Public Member Functions

- [ovector_const_reverse_tlate](#) (const [ovector_const_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).

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

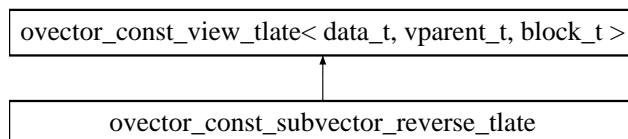
- [ovector_rev_tlate.h](#)

8.306 ovector_const_subvector_reverse_tlate Class Template Reference

Reversed view of a const subvector.

```
#include <ovector_rev_tlate.h>
```

Inheritance diagram for ovector_const_subvector_reverse_tlate::



8.306.1 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.

Warning:

At present, reversing a reversed vector does not give the original ordering.

Definition at line 497 of file ovector_rev_tlate.h.

Public Member Functions

- [ovector_const_subvector_reverse_tlate](#) (const [ovector_const_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).

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

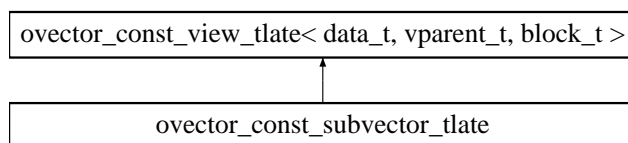
- [ovector_rev_tlate.h](#)

8.307 `ovector_const_subvector_tlate` Class Template Reference

Create a const vector from a subvector of another vector.

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ovector_const_subvector_tlate`:



8.307.1 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 1576 of file `ovector_tlate.h`.

Public Member Functions

- [ovector_const_subvector_tlate](#) (const [ovector_const_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:

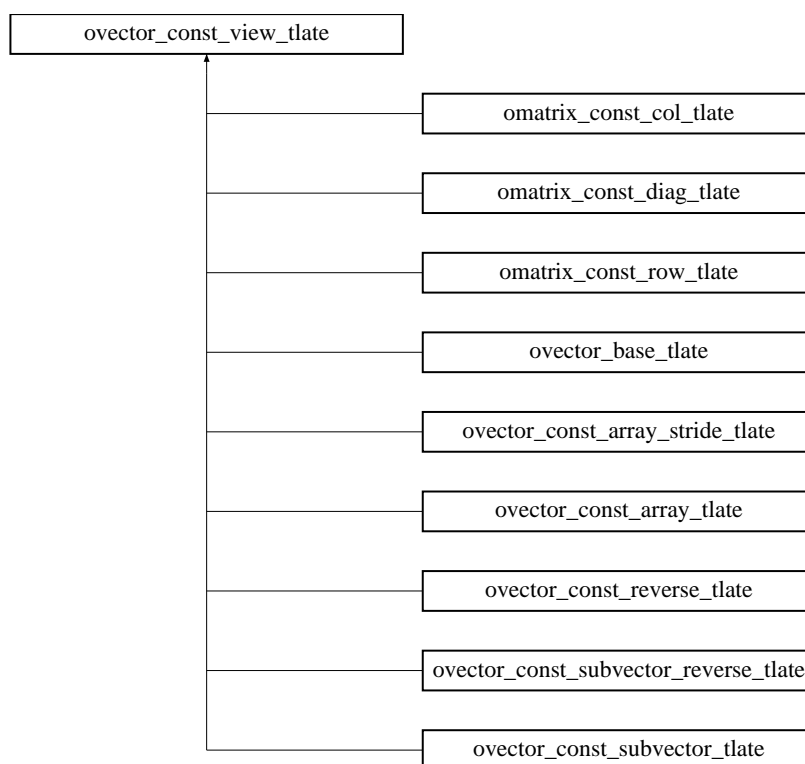
- [ovector_tlate.h](#)

8.308 `ovector_const_view_tlate` Class Template Reference

A const vector view with finite stride.

```
#include <ovector_tlate.h>
```

Inheritance diagram for `ovector_const_view_tlate`:



8.308.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class `ovector_const_view_tlate`< data_t, vparent_t, block_t >

A const vector view with finite stride.

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

Public Member Functions

Copy constructors

- [ovector_const_view_tlate](#) (const [ovector_const_view_tlate](#) &v)
Shallow copy constructor - create a new view of the same vector.
- [ovector_const_view_tlate](#) & [operator=](#) (const [ovector_const_view_tlate](#) &v)
Shallow copy constructor - create a new view of the same vector.
- [ovector_const_view_tlate](#) (const [uvector_const_view_tlate](#)< data_t > &v)
Shallow copy constructor - view a unit-stride vector.
- [ovector_const_view_tlate](#) & [operator=](#) (const [uvector_const_view_tlate](#)< data_t > &v)
Shallow copy constructor - view a unit-stride vector.

Get methods

- const data_t & [operator\[\]](#) (size_t i) const
Array-like indexing.
- const data_t & [operator\(\)](#) (size_t i) const
Array-like indexing with operator().
- 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).

- `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.

Other methods

- `data_t norm () const`
Norm.
- `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 vector for a particular value and return the closest match.
- `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.

Protected Member Functions

- `ovector_const_view_tlate ()`
Empty constructor provided for use by `ovector_view_tlate(const ovector_view_tlate &v)`.

8.308.2 Member Function Documentation

8.308.2.1 `size_t capacity () const` [inline]

Method to return capacity.

Analogous to `std::vector<>.capacity()`.

Definition at line 193 of file `ovector_tlate.h`.

8.308.2.2 `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 "vector". If `is_owner()` is true, then it is an `ovector_tlate` object.

If any O₂sc1 class creates a `ovector_tlate` object in which `is_owner()` returns false, then it is a bug and should be reported.

Definition at line 262 of file `ovector_tlate.h`.

8.308.2.3 `size_t lookup (const data_t x0) const` [inline]

Exhaustively look through the vector for a particular value and return the closest match.

This can only fail if the vector is empty or if *all* of the entries in the vector are not finite. In these cases the function calls the error handler and returns 0.

If more than one entry is the same distance from `x0`, this function returns the entry with smallest index.

Definition at line 277 of file `ovector_tlate.h`.

8.308.2.4 data_t max () const [inline]

Find the maximum element.

This can only fail if *all* of the entries in the array are not finite or if the vector is empty, in which case it calls the error handler and returns 0.

Definition at line 309 of file ovector_tlate.h.

8.308.2.5 size_t max_index () const [inline]

Find the location of the maximum element.

This can only fail if *all* of the entries in the array are not finite or if the vector is empty, in which case it calls the error handler and returns 0.

Definition at line 344 of file ovector_tlate.h.

8.308.2.6 data_t min () const [inline]

Find the minimum element.

This can only fail if *all* of the entries in the array are not finite or if the vector is empty, in which case it calls the error handler and returns 0.

Definition at line 382 of file ovector_tlate.h.

8.308.2.7 size_t min_index () const [inline]

Find the location of the minimum element.

This can only fail if *all* of the entries in the array are not finite or if the vector is empty, in which case it calls the error handler and returns 0.

Definition at line 417 of file ovector_tlate.h.

8.308.2.8 data_t norm () const [inline]

Norm.

For an empty vector, this function returns zero and does not call the error handler.

Idea for future

Move this function outside the vector template since it doesn't really work with integers.

Definition at line 220 of file ovector_tlate.h.

8.308.2.9 size_t size () const [inline]

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 184 of file ovector_tlate.h.

8.308.2.10 size_t stride () const [inline]

Method to return vector stride.

If no memory has been allocated, this will quietly return zero.

Definition at line 204 of file `ovector_tlate.h`.

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

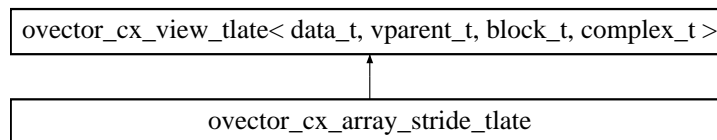
- [ovector_tlate.h](#)

8.309 `ovector_cx_array_stride_tlate` Class Template Reference

Create a vector from an array with a stride.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_array_stride_tlate`:



8.309.1 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 752 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:

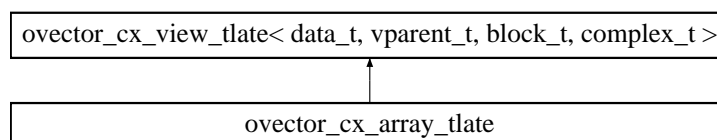
- [ovector_cx_tlate.h](#)

8.310 `ovector_cx_array_tlate` Class Template Reference

Create a vector from an array.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_array_tlate`:



8.310.1 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 732 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:

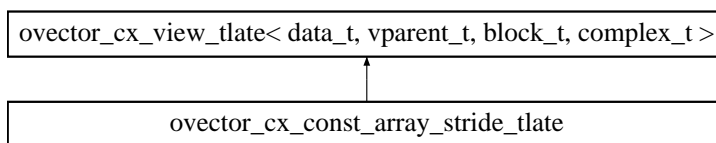
- [ovector_cx_tlate.h](#)

8.311 ovector_cx_const_array_stride_tlate Class Template Reference

Create a vector from an array_stride.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_const_array_stride_tlate`:



8.311.1 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 854 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 [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:

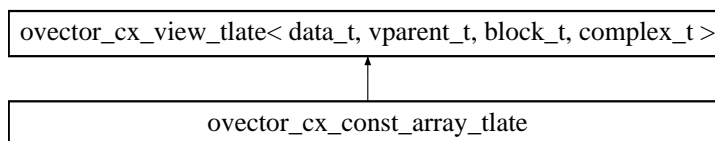
- [ovector_cx_tlate.h](#)

8.312 ovector_cx_const_array_tlate Class Template Reference

Create a vector from an array.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_const_array_tlate::



8.312.1 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 802 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.

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 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:

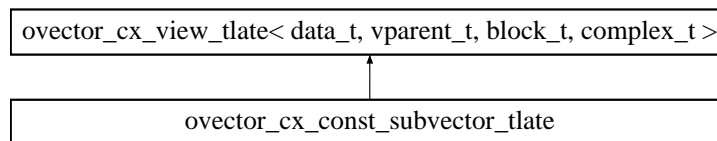
- [ovector_cx_tlate.h](#)

8.313 `ovector_cx_const_subvector_tlate` Class Template Reference

Create a vector from a subvector of another.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_const_subvector_tlate`:



8.313.1 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 907 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 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:

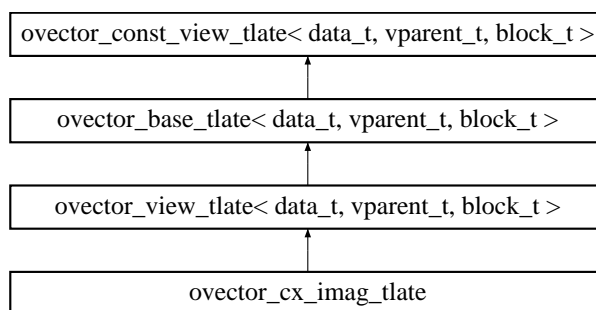
- [ovector_cx_tlate.h](#)

8.314 ovector_cx_imag_tlate Class Template Reference

Create a imaginary vector from the imaginary parts of a complex vector.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_imag_tlate::



8.314.1 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 986 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:

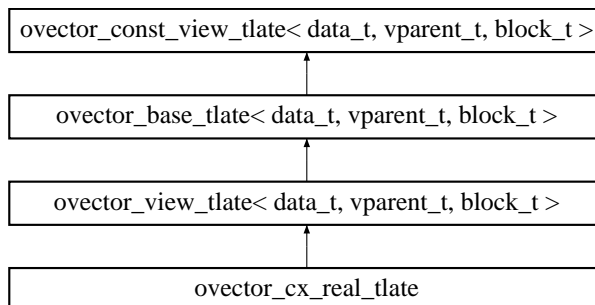
- [ovector_cx_tlate.h](#)

8.315 ovector_cx_real_tlate Class Template Reference

Create a real vector from the real parts of a complex vector.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_real_tlate::



8.315.1 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 964 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:

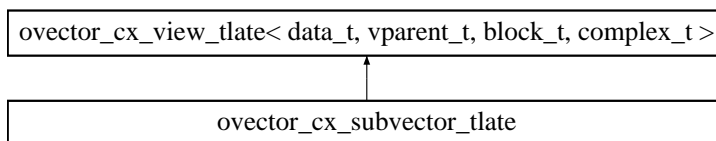
- [ovector_cx_tlate.h](#)

8.316 ovector_cx_subvector_tlate Class Template Reference

Create a vector from a subvector of another.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_subvector_tlate::



8.316.1 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 773 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:

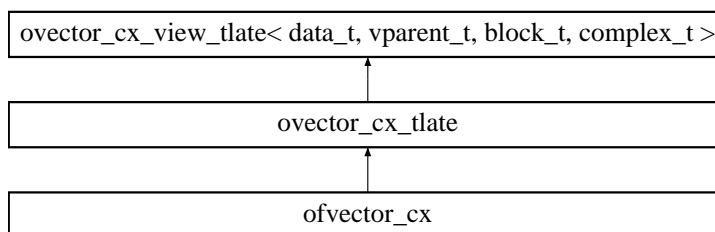
- [ovector_cx_tlate.h](#)

8.317 ovector_cx_tlate Class Template Reference

A vector of double-precision numbers.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_tlate::



8.317.1 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

Add subvector_stride, const_subvector_stride

Definition at line 489 of file ovector_cx_tlate.h.

Public Member Functions

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.

8.317.2 Member Function Documentation

8.317.2.1 `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 702 of file `ovector_cx_tlate.h`.

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

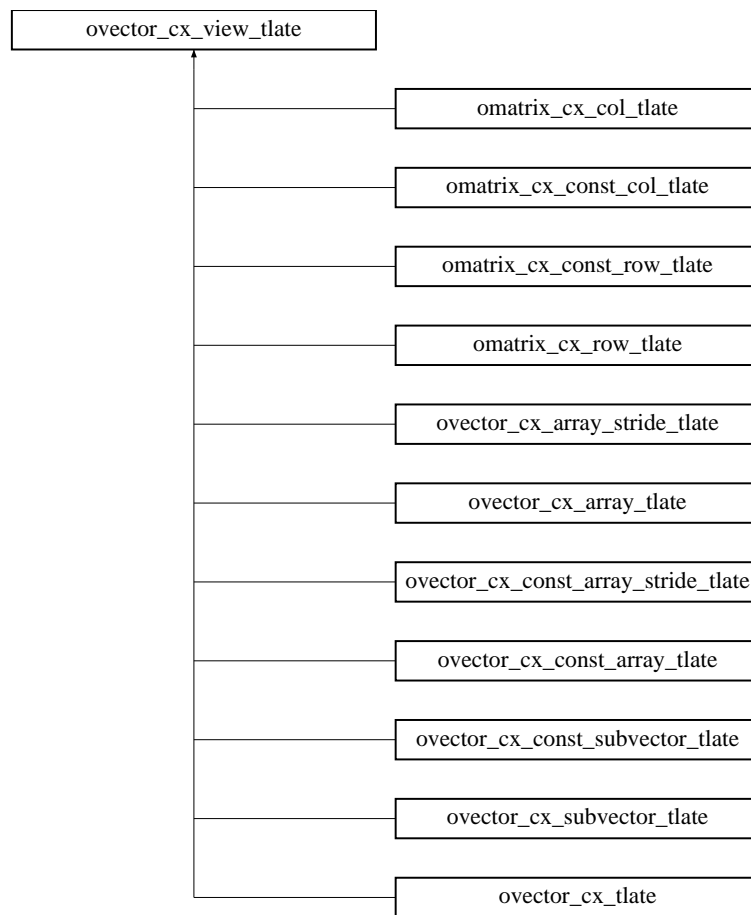
- [ovector_cx_tlate.h](#)

8.318 `ovector_cx_view_tlate` Class Template Reference

A vector view of double-precision numbers.

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for `ovector_cx_view_tlate::`



8.318.1 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.

Definition at line 53 of file `ovector_cx_tlate.h`.

Public Member Functions

- `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)`
Shallow copy constructor - create a new view of the same vector.
- `ovector_cx_view_tlate & operator= (const ovector_cx_view_tlate &v)`
Shallow 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].

8.318.2 Member Function Documentation

8.318.2.1 `size_t size () const` [inline]

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 310 of file `ovector_cx_tlate.h`.

8.318.2.2 `size_t stride () const` [inline]

Method to return vector stride.

If no memory has been allocated, this will quietly return zero.

Definition at line 320 of file `ovector_cx_tlate.h`.

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

- [ovector_cx_tlate.h](#)

8.319 `ovector_int_alloc` Class Reference

A simple class to provide an `allocate ()` function for `ovector_int`.

```
#include <ovector_tlate.h>
```

8.319.1 Detailed Description

A simple class to provide an `allocate ()` function for `ovector_int`.

Definition at line 1702 of file `ovector_tlate.h`.

Public Member Functions

- void `allocate (ovector_int &o, size_t i)`
Allocate \forall for i elements.
- void `free (ovector_int &o)`
Free memory.

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

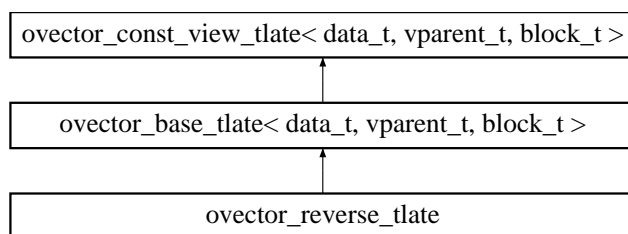
- [ovector_tlate.h](#)

8.320 `ovector_reverse_tlate` Class Template Reference

Reversed view of a vector.

```
#include <ovector_rev_tlate.h>
```

Inheritance diagram for `ovector_reverse_tlate::`



8.320.1 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:

Note that you can't reverse a reversed vector, and this is why this class does not have a constructor of the form `ovector_reverse(ovector_reverse &v)`.

Definition at line 56 of file `ovector_rev_tlate.h`.

Public Member Functions

- [ovector_reverse_tlate](#) ([ovector_tlate](#)< data_t, vparent_t, block_t > &v)
Create a vector from dat with size n and stride s.
- [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:

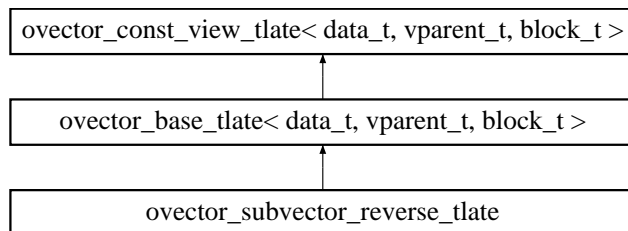
- [ovector_rev_tlate.h](#)

8.321 ovector_subvector_reverse_tlate Class Template Reference

Reversed view of a subvector.

```
#include <ovector_rev_tlate.h>
```

Inheritance diagram for ovector_subvector_reverse_tlate::



8.321.1 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.

Warning:

At present, reversing a reversed vector does not give the original ordering.

Definition at line 326 of file `ovector_rev_tlate.h`.

Public Member Functions

- [ovector_subvector_reverse_tlate](#) ([ovector_base_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:

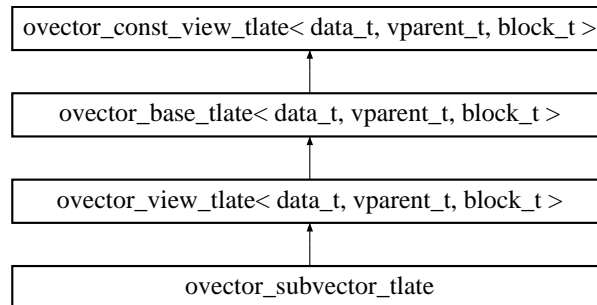
- [ovector_rev_tlate.h](#)

8.322 ovector_subvector_tlate Class Template Reference

Create a vector from a subvector of another.

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_subvector_tlate::



8.322.1 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.

The constructor will fail if the original vector is empty, or if the user requests a subvector which includes elements beyond the end of the original vector.

Definition at line 1473 of file ovector_tlate.h.

Public Member Functions

- [ovector_subvector_tlate](#) ([ovector_base_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:

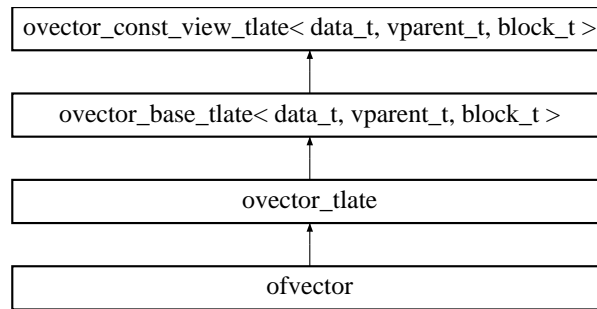
- [ovector_tlate.h](#)

8.323 ovector_tlate Class Template Reference

A vector with finite stride.

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_tlate::



8.323.1 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`. They are documented in the "Functions" section of `ovector_tlate.h`.

Design notes

At present, the owner variable can be used to distinguish between ovector and ovector_views: for views, owner is always zero, but for normal ovector, owner is 1 (even for empty vectors).

The `reserve()`, `pop_back()`, and `push_back()` methods require the ability to have empty vectors which still have memory allocated for them, so the proper test for whether or not memory is allocated is to test whether or not 'block' is zero. This means we should never have a case where the block is non-zero (i.e. there is memory allocated there) but the size of the block is zero. This is the test used in the destructor and the `free()` method. The `free` method also sets block to zero accordingly.

Definition at line 922 of file `ovector_tlate.h`.

Public Member Functions

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.
- `template<class alt_vec_t > ovector_tlate` (size_t nv, alt_vec_t &v)
Deep copy constructor for generic vectors.
- `ovector_tlate` (const `ovector_const_view_tlate`< data_t, vparent_t, block_t > &v)
Deep copy constructor for other related vectors.
- `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_const_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 `ovector_const_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)

- `int free ()`
*Allocate memory for size `n` after freeing any memory currently in use.
Free the memory.*

Stack-like operations

- `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_back ()`
Return the last value and shrink the vector size by one.

Other methods

- `int erase (size_t ix)`
Remove element with index `ix` and decrease the vector size by one.
- `int sort_unique ()`
Sort the vector and ensure all elements are unique by removing duplicates.

Protected Member Functions

- `void intl_sanity_check (size_t ix)`
- `int intl_allocate (size_t nsize)`
An internal allocate function.
- `int intl_free ()`
The internal free function.
- `template<class alt_vec_t >`
`int intl_init (size_t n, alt_vec_t &v)`
An internal init() function.
- `template<class alt_vec_t >`
`ovector_tlate & intl_assign (size_t n, alt_vec_t &v)`
An internal assignment function for `operator=()`.

8.323.2 Member Function Documentation

8.323.2.1 int allocate (size_t nsize) [inline]

Allocate memory for size `n` after freeing any memory currently in use.

Note that this automatically deallocates any previously allocated memory before attempting to allocate more. If this allocation fails (i.e. because we ran out of memory) then the original vector will still have been deallocated.

Definition at line 1181 of file `ovector_tlate.h`.

8.323.2.2 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 1195 of file `ovector_tlate.h`.

8.323.2.3 int intl_allocate (size_t nsize) [inline, protected]

An internal allocate function.

Note:

This function does nothing if `nsize` is zero. Also, this does not free any already allocated memory first (unlike the user-interface version `allocate()`).

Definition at line 949 of file ovector_tlate.h.

8.323.2.4 int intl_init (size_t n, alt_vec_t & v) [inline, protected]

An internal init() function.

This function calls [intl_allocate\(\)](#) first, and then copies the data from the vector v.

Definition at line 1014 of file ovector_tlate.h.

8.323.2.5 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.

If the argument `cap` is smaller than the present vector size given by [size\(\)](#), then this function does nothing and the error handler is not called.

Definition at line 1268 of file ovector_tlate.h.

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

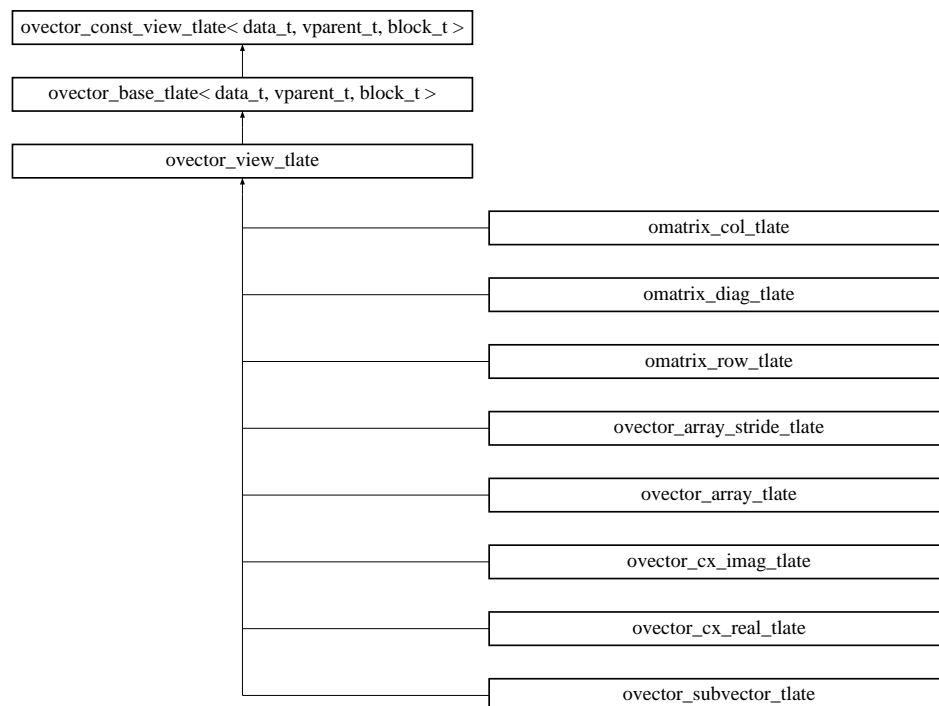
- [ovector_tlate.h](#)

8.324 ovector_view_tlate Class Template Reference

A vector view with finite stride.

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_view_tlate::



8.324.1 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 722 of file ovector_tlate.h.

Public Member Functions

Copy constructors

- [ovector_view_tlate](#) (const [ovector_view_tlate](#) &v)
Shallow copy constructor - create a new view of the same vector.
- [ovector_view_tlate](#) & [operator=](#) (const [ovector_view_tlate](#) &v)
Shallow copy constructor - create a new view of the same vector.
- [ovector_view_tlate](#) ([ovector_base_tlate](#)< data_t, vparent_t, block_t > &v)
Shallow copy constructor for non-views.
- [ovector_view_tlate](#) & [operator=](#) ([ovector_base_tlate](#)< data_t, vparent_t, block_t > &v)
Shallow copy constructor for non-views.
- [ovector_view_tlate](#) ([uvector_base_tlate](#)< data_t > &v)
Shallow copy constructor for unit-stride vectors.
- [ovector_view_tlate](#) & [operator=](#) ([uvector_base_tlate](#)< data_t > &v)
Shallow copy constructor for unit-stride vectors.

Get and set methods

- data_t & [operator\[\]](#) (size_t i) const
Array-like indexing.
- data_t & [operator\(\)](#) (size_t i) const
Array-like indexing with operator().
- data_t * [get_ptr](#) (size_t i) const
Get pointer (with optional range-checking).
- int [set](#) (size_t i, data_t val) const
Set (with optional range-checking).
- int [set_all](#) (double val) const
Set all of the value to be the value val.

Protected Member Functions

- [ovector_view_tlate](#) ()
Empty constructor provided for use by [ovector_view_tlate\(const ovector_view_tlate &v\)](#).

8.324.2 Member Function Documentation

8.324.2.1 int set_all (double val) const [inline]

Set all of the value to be the value val.

If the vector is empty, this function does not perform any assignment and does not call the error handler.

Definition at line 876 of file ovector_tlate.h.

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

- [ovector_tlate.h](#)

8.325 permutation Class Reference

A [permutation](#).

```
#include <permutation.h>
```

8.325.1 Detailed Description

A [permutation](#).

This [permutation](#) class is completely compatible with the GSL [permutation](#) object.

Definition at line 73 of file permutation.h.

Public Member Functions

- **permutation** (size_t dim=0)
- size_t & **operator[]** (size_t i)
Array-like indexing.
- const size_t & **operator[]** (size_t i) const
Array-like indexing.
- size_t & **operator()** (size_t i)
Array-like indexing.
- const size_t & **operator()** (size_t i) const
Array-like indexing.
- size_t **get** (size_t i) const
Get (with optional range-checking).
- int **set** (size_t i, size_t val)
Set (with optional range-checking).
- int **init** ()
Initialize [permutation](#) to the identity.
- size_t **size** () const
Return [permutation](#) size.
- int **allocate** (size_t dim)
Allocate memory for a [permutation](#) of size dim.
- int **free** ()
Free the memory.
- int **swap** (const size_t i, const size_t j)
Swap two elements of a [permutation](#).
- bool **valid** () const
Check to see that a [permutation](#) is valid.
- int **reverse** ()
Reverse the [permutation](#).
- **permutation inverse** () const
Compute the inverse of a [permutation](#).
- template<class vec_t >
int **apply** (vec_t &v) const
Apply the [permutation](#) to a vector.
- template<class vec_t >
int **apply_inverse** (vec_t &v) const
Apply the inverse [permutation](#) to a vector.

Copy constructors

- **permutation** (const [permutation](#) &v)
- [permutation](#) & **operator=** (const [permutation](#) &v)

8.325.2 Member Function Documentation

8.325.2.1 `int apply (vec_t & v) const` [inline]

Apply the [permutation](#) to a vector.

Now have $k=i$, i.e. the least in its cycle

Definition at line 300 of file permutation.h.

8.325.2.2 `int apply_inverse (vec_t & v) const` [inline]

Apply the inverse [permutation](#) to a vector.

Now have $k=i$, i.e. the least in its cycle

Definition at line 325 of file permutation.h.

8.325.2.3 `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 249 of file permutation.h.

8.325.2.4 `size_t size () const` [inline]

Return [permutation](#) size.

If no memory has been allocated, this will quietly return zero.

Definition at line 227 of file permutation.h.

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

- [permutation.h](#)

8.326 pinside Class Reference

Test [line](#) intersection and [point](#) inside polygon.

```
#include <pinside.h>
```

8.326.1 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.

Idea for future

The [inside\(\)](#) functions actually copy the points twice. This can be made more efficient.

Definition at line 67 of file pinside.h.

Data Structures

- struct [line](#)
Internal [line](#) definition for [pinside](#).
- struct [point](#)
Internal [point](#) definition for [pinside](#).

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_base](#) &xa, const [o2scl::ovector_base](#) &ya)
Determine if [point](#) (x,y) is inside a polygon.
- template<class vec_t >
int [inside](#) (double x, double y, size_t n, const vec_t &xa, const vec_t &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.

8.326.2 Member Function Documentation

8.326.2.1 int inside (double x, double y, size_t n, const vec_t & xa, const vec_t & ya) [inline]

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.

The parameter n should be the number of polygon points specified in vectors xa and ya.

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.

Definition at line 134 of file pinside.h.

8.326.2.2 int inside (double x, double y, const o2scl::ovector_base & xa, const o2scl::ovector_base & 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.

8.326.2.3 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 100 of file pinside.h.

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

- pinside.h

8.327 pinside::line Struct Reference

Internal [line](#) definition for [pinside](#).

```
#include <pinside.h>
```

8.327.1 Detailed Description

Internal [line](#) definition for [pinside](#).

Definition at line 78 of file pinside.h.

Data Fields

- [point](#) p1
- [point](#) p2

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

- pinside.h

8.328 pinside::point Struct Reference

Internal [point](#) definition for [pinside](#).

```
#include <pinside.h>
```

8.328.1 Detailed Description

Internal [point](#) definition for [pinside](#).

Definition at line 72 of file pinside.h.

Data Fields

- double x
- double y

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

- pinside.h

8.329 planar_intp Class Template Reference

Interpolate among two independent variables with planes.

```
#include <planar_intp.h>
```

8.329.1 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 $x_i, y_i, f_{j,i}$, the values of 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. Distances are determined with

$$d_{ij} = \sqrt{\left(\frac{x_i - x_j}{\Delta x}\right)^2 + \left(\frac{y_i - y_j}{\Delta y}\right)^2}$$

where $\Delta x = x_{\max} - x_{\min}$ and $\Delta y = y_{\max} - y_{\min}$.

This procedure will fail if the three closest points are co-linear, and `interp()` will then call `O2SCL_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[]`.

For example, with 10 points in the x - y plane with $-1 < x < 1$ and $-1 < y < 1$, the planes are demarcated according to

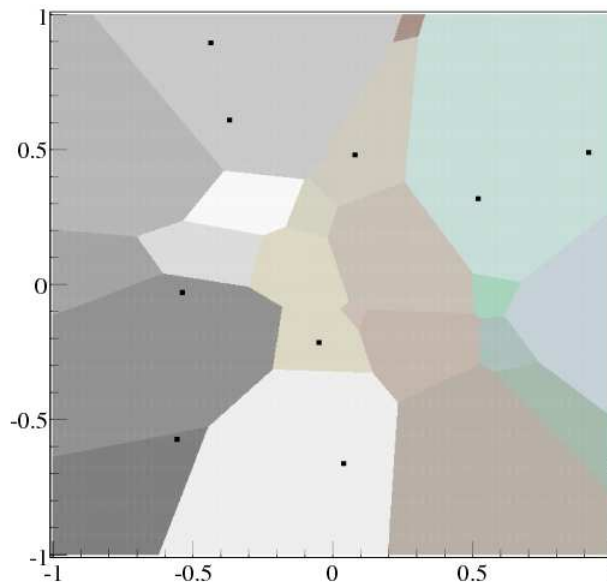


Figure 2: in_planar_intp2.eps

where each polygonal region represents the set of all points in the domain which will be mapped to the same plane to approximate the function.

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.

Idea for future

Instead of comparing `den` with zero, add the option of comparing it with a small number.

Definition at line 77 of file `planar_intp.h`.

Public Member Functions

- 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, size_t &i1, double &x1, double &y1, size_t &i2, double &x2, double &y2, size_t &i3, double &x3, double &y3)
Planar interpolation returning the closest points.

Protected Member Functions

- int [swap](#) (size_t &i1, double &c1, size_t &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.

8.329.2 Member Function Documentation

8.329.2.1 int [interp](#) (double x, double y, vec_t &ip, size_t &i1, double &x1, double &y1, size_t &i2, double &x2, double &y2, size_t &i3, 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 121 of file planar_intp.h.

8.329.2.2 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 106 of file planar_intp.h.

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

- planar_intp.h

8.330 `pointer_2d_alloc` Class Template Reference

A simple class to provide an `allocate()` function for pointers.

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

8.330.1 Detailed Description

```
template<class base_t> class pointer_2d_alloc< base_t >
```

A simple class to provide an `allocate()` function for pointers.

Uses `new` and `delete`.

Definition at line 151 of file `array.h`.

Public Member Functions

- void `allocate` (base_t **&v, size_t i, size_t j)
Allocate \forall for i elements.
- void `free` (base_t **&v, size_t i)
Free memory.

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

- [array.h](#)

8.331 `pointer_alloc` Class Template Reference

A simple class to provide an `allocate()` function for pointers.

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

8.331.1 Detailed Description

```
template<class base_t> class pointer_alloc< base_t >
```

A simple class to provide an `allocate()` function for pointers.

This class is used in `tensor_grid`.

This class uses `new` and `delete`.

Todo

Call error handler appropriately here

Definition at line 137 of file `array.h`.

Public Member Functions

- void `allocate` (base_t *&v, size_t i)
Allocate \forall for i elements.
 - void `free` (base_t *&v)
Free memory.
-

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

- [array.h](#)

8.332 `pointer_input` Struct Reference

A pointer input structure.

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

8.332.1 Detailed Description

A pointer input structure.

This class is experimental.

Definition at line 103 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](#)

8.333 `pointer_output` Struct Reference

A pointer output structure.

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

8.333.1 Detailed Description

A pointer output structure.

This class is experimental.

Definition at line 90 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:

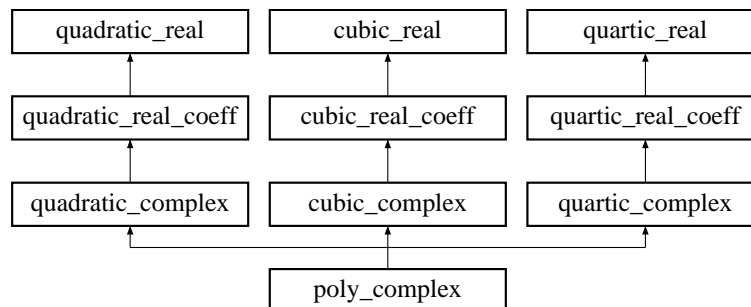
- [collection.h](#)

8.334 poly_complex Class Reference

Solve a general polynomial with complex coefficients [abstract base].

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

Inheritance diagram for poly_complex::



8.334.1 Detailed Description

Solve a general polynomial with complex coefficients [abstract base].

Definition at line 360 of file poly.h.

Public Member Functions

- virtual int [solve_c](#) (int n, const std::complex< double > co[], std::complex< double > ro[])=0
Solve the n-th order polynomial.
- virtual int [polish_c](#) (int n, const std::complex< double > co[], std::complex< double > *ro)=0
Polish the roots.
- const char * [type](#) ()
Return a string denoting the type ("poly_complex").

8.334.2 Member Function Documentation

8.334.2.1 virtual int solve_c(int n, const std::complex< double > co[], std::complex< double > ro[]) [pure 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].

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

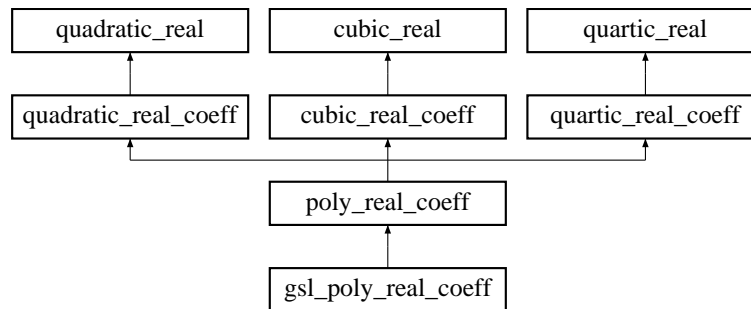
- [poly.h](#)

8.335 poly_real_coeff Class Reference

Solve a general polynomial with real coefficients and complex roots [abstract base].

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

Inheritance diagram for poly_real_coeff::



8.335.1 Detailed Description

Solve a general polynomial with real coefficients and complex roots [abstract base].

Definition at line 336 of file poly.h.

Public Member Functions

- virtual int [solve_rc](#) (int n, const double co[], std::complex< double > ro[]) = 0
Solve the n-th order polynomial.
- const char * [type](#) ()
Return a string denoting the type ("poly_real_coeff").

8.335.2 Member Function Documentation

8.335.2.1 virtual int solve_rc (int n, const double co[], std::complex< double > ro[]) [pure 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].

Implemented in [gsl_poly_real_coeff](#).

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

- [poly.h](#)

8.336 polylog Class Reference

Polylogarithms (approximate) $Li_n(x)$.

```
#include <polylog.h>
```

8.336.1 Detailed Description

Polylogarithms (approximate) $Li_n(x)$.

This class is experimental.

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

$$\begin{aligned} Li_2\left(\frac{1}{2}\right) &= \frac{1}{12} \left[\pi^2 - 6 (\ln 2)^2 \right] \\ Li_3\left(\frac{1}{2}\right) &= \frac{1}{24} \left[4 (\ln 2)^3 - 2\pi^2 \ln 2 + 21\zeta(3) \right] \\ Li_{-1}(x) &= \frac{x}{(1-x)^2} \\ Li_{-2}(x) &= \frac{x(x+1)}{(1-x)^3} \end{aligned}$$

Definition at line 79 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:

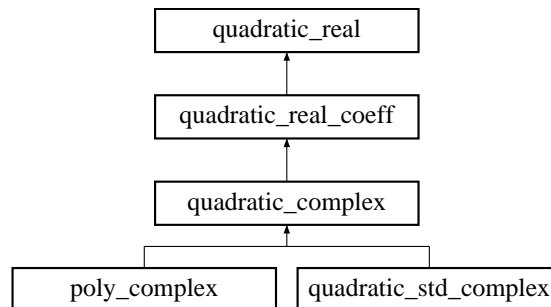
- polylog.h

8.337 quadratic_complex Class Reference

Solve a quadratic polynomial with complex coefficients and complex roots [abstract base].

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

Inheritance diagram for quadratic_complex::



8.337.1 Detailed Description

Solve a quadratic polynomial with complex coefficients and complex roots [abstract base].

Definition at line 104 of file poly.h.

Public Member Functions

- 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)=0
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:

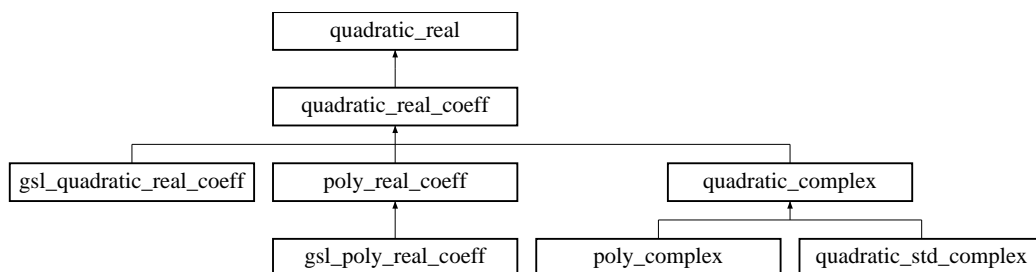
- [poly.h](#)

8.338 quadratic_real Class Reference

Solve a quadratic polynomial with real coefficients and real roots [abstract base].

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

Inheritance diagram for quadratic_real::



8.338.1 Detailed Description

Solve a quadratic polynomial with real coefficients and real roots [abstract base].

Definition at line 58 of file poly.h.

Public Member Functions

- virtual int [solve_r](#) (const double a2, const double b2, const double c2, double &x1, double &x2)=0
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:

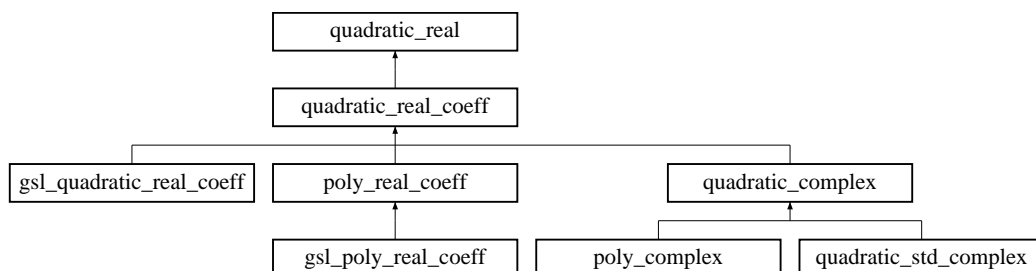
- [poly.h](#)

8.339 quadratic_real_coeff Class Reference

Solve a quadratic polynomial with real coefficients and complex roots [abstract base].

#include <poly.h>

Inheritance diagram for quadratic_real_coeff::



8.339.1 Detailed Description

Solve a quadratic polynomial with real coefficients and complex roots [abstract base].

Definition at line 77 of file poly.h.

Public Member Functions

- 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)=0
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:

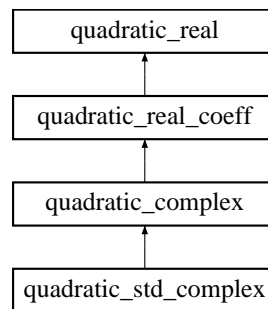
- [poly.h](#)

8.340 quadratic_std_complex Class Reference

Solve a quadratic with complex coefficients and complex roots.

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

Inheritance diagram for quadratic_std_complex::



8.340.1 Detailed Description

Solve a quadratic with complex coefficients and complex roots.

Definition at line 651 of file poly.h.

Public Member Functions

- 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:

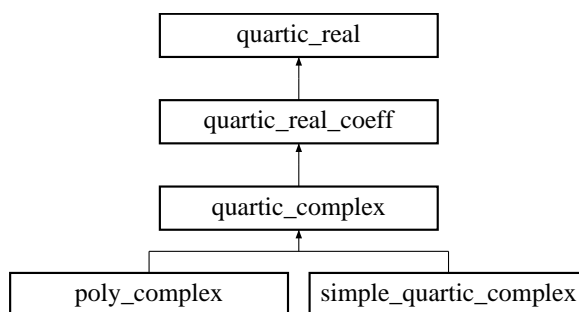
- [poly.h](#)

8.341 quartic_complex Class Reference

Solve a quartic polynomial with complex coefficients and complex roots [abstract base].

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

Inheritance diagram for quartic_complex::



8.341.1 Detailed Description

Solve a quartic polynomial with complex coefficients and complex roots [abstract base].

Definition at line 289 of file poly.h.

Public Member Functions

- 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 $x = x_4$.
- 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 $x = x_1$, $x = x_2$, $x = x_3$, and $x = x_4$.
- 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)=0
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 $x = x_4$.
- const char * [type](#) ()
Return a string denoting the type ("quartic_complex").

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

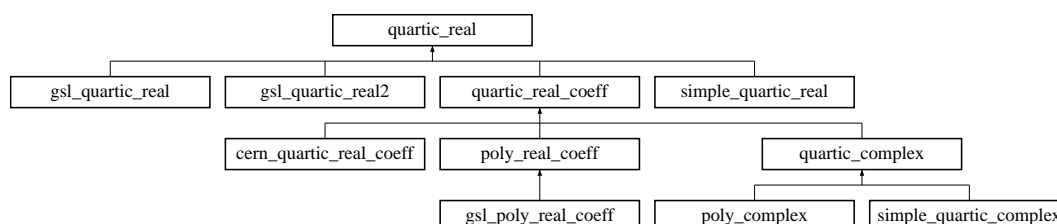
- [poly.h](#)

8.342 quartic_real Class Reference

Solve a quartic polynomial with real coefficients and real roots [abstract base].

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

Inheritance diagram for quartic_real::



8.342.1 Detailed Description

Solve a quartic polynomial with real coefficients and real roots [abstract base].

Definition at line 231 of file `poly.h`.

Public Member Functions

- 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)=0
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 ("quartic_real").

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

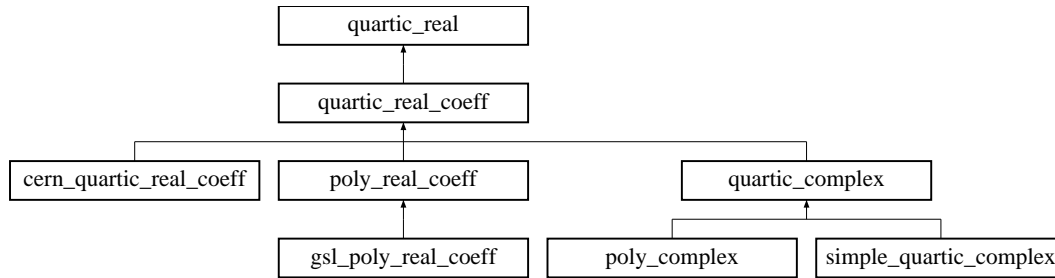
- `poly.h`

8.343 `quartic_real_coeff` Class Reference

Solve a quartic polynomial with real coefficients and complex roots [abstract base].

`#include <poly.h>`

Inheritance diagram for `quartic_real_coeff`:



8.343.1 Detailed Description

Solve a quartic polynomial with real coefficients and complex roots [abstract base].

Definition at line 255 of file `poly.h`.

Public Member Functions

- 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 $x = x_4$.
- 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)=0
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 $x = x_4$.
- const char * `type` ()
Return a string denoting the type ("quartic_real_coeff").

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

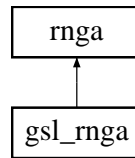
- `poly.h`

8.344 rnga Class Reference

Random number generator base.

```
#include <rnga.h>
```

Inheritance diagram for rnga::



8.344.1 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()`.

Idea for future

Consider some analog of the GSL function `gsl_rng_uniform_pos()`, i.e. as used in the GSL Monte Carlo classes.

Definition at line 54 of file `rnga.h`.

Public Member Functions

- `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.

8.344.2 Member Function Documentation

8.344.2.1 void clock_seed ()

Initialize the seed with a value taken from the computer clock.

This is a naive seed generator which uses `seed=time (0)` to generate a seed.

Idea for future

Figure out a better way of computing a random seed in a platform-independent way.

Reimplemented in [gsl_rnga](#).

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

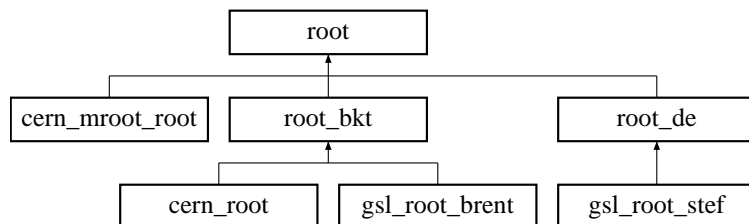
- [rnga.h](#)

8.345 root Class Template Reference

1-dimensional solver [abstract base]

```
#include <root.h>
```

Inheritance diagram for root::



8.345.1 Detailed Description

```
template<class param_t, class func_t = funct<param_t>, class dfunc_t = func_t> class root< param_t, func_t, dfunc_t >
```

1-dimensional solver [abstract base]

Idea for future

Maybe consider allowing the user to specify the stream to which 'verbose' information is sent.

Definition at line 43 of file root.h.

Public Member Functions

- 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)=0
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 [err_nonconv](#)
If true, call the error handler if the routine does not "converge".
- int [last_conv](#)
Zero if last call to [solve\(\)](#), [solve_bkt\(\)](#), or [solve_de\(\)](#) converged.
- int [last_ntrial](#)
The number of iterations for in the most recent minimization.

8.345.2 Member Function Documentation

8.345.2.1 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 105 of file root.h.

8.345.3 Field Documentation

8.345.3.1 int last_conv

Zero if last call to [solve\(\)](#), [solve_bkt\(\)](#), or [solve_de\(\)](#) converged.

This is particularly useful if `err_nonconv` is false to test if the last call to [solve\(\)](#), [solve_bkt\(\)](#), or [solve_de\(\)](#) converged.

Definition at line 85 of file root.h.

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

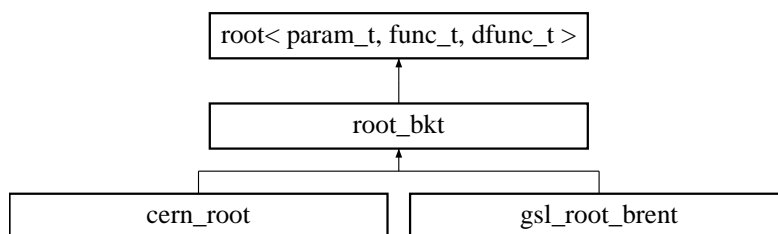
- root.h

8.346 root_bkt Class Template Reference

1-dimensional bracketing solver [abstract base]

```
#include <root.h>
```

Inheritance diagram for root_bkt::



8.346.1 Detailed Description

`template<class param_t, class func_t = funct<param_t>, class dfunc_t = func_t> class root_bkt< param_t, func_t, dfunc_t >`

1-dimensional bracketing solver [abstract base]

Definition at line 154 of file root.h.

Public Member Functions

- virtual const char * [type](#) ()
Return the type, "root_bkt".
- virtual int [solve_bkt](#) (double &x1, double x2, param_t &pa, func_t &func)=0
Solve func in region $x_1 < x < x_2$ returning x_1 .
- virtual int [solve](#) (double &x, param_t &pa, func_t &func)
Solve func using x as an initial guess.
- 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 [bracket_step](#)
The stepsize for automatic bracketing (default 10^{-4}).

8.346.2 Field Documentation

8.346.2.1 double bracket_step

The stepsize for automatic bracketing (default 10^{-4}).

If this is exactly zero, it will be reset to 10^{-4} by [solve\(\)](#).

Definition at line 172 of file root.h.

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

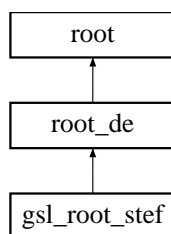
- root.h

8.347 root_de Class Template Reference

1-dimensional with solver with derivatives [abstract base]

```
#include <root.h>
```

Inheritance diagram for root_de::



8.347.1 Detailed Description

template<class param_t, class func_t = [funct](#)<param_t>, class dfunc_t = func_t> class [root_de](#)< param_t, func_t, dfunc_t >

1-dimensional with solver with derivatives [abstract base]

Idea for future

At the moment, the functions [solve\(\)](#) and [solve_bkt\(\)](#) are not implemented for derivative solvers.

Definition at line 267 of file root.h.

Public Member Functions

- virtual const char * [type](#) ()
Return the type, "root_de".
- 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](#) (double &x, param_t &pa, func_t &func)
Solve func using x as an initial guess.
- virtual int [solve_de](#) (double &x, param_t &pa, func_t &func, dfunc_t &df)=0
Solve func using x as an initial guess using derivatives df.

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

- root.h

8.348 search_vec Class Template Reference

Searching class for monotonic data with caching.

```
#include <search_vec.h>
```

8.348.1 Detailed Description

template<class vec_t> class [search_vec](#)< vec_t >

Searching class for monotonic data with caching.

A searching class for monotonic vectors. A caching system similar to [gsl_interp_accel](#) is used.

To find the interval containing a value, use [find_interval\(\)](#). If you happen to know in advance that the vector is increasing or decreasing, then you can use [find_interval_inc\(\)](#) or [find_interval_dec\(\)](#) instead. To ignore the caching and just use straight binary search, you can use [bsearch_inc\(\)](#) or [bsearch_dec\(\)](#) for increasing or decreasing arrays respectively.

If you want to allow the function to return $n-1$, to indicate the value is larger than or equal to the last element, use `find_interval_uncons()`.

Alternatively, if you just want to find the index with the element closest to a specified value, use `ordered_lookup()`. Note that `ordered_lookup()` is slightly different than `ovector_tlate::lookup()`, since the latter does not assume the data is monotonic.

Definition at line 69 of file `search_vec.h`.

Public Member Functions

- `size_t find_interval` (const double `x0`, `size_t` `n`, const `vec_t` &`x`)
Search an increasing or decreasing vector for the interval containing `x0`.
- `size_t find_interval_inc` (const double `x0`, `size_t` `n`, const `vec_t` &`x`)
Search an increasing vector for the interval containing `x0`.
- `size_t find_interval_dec` (const double `x0`, `size_t` `n`, const `vec_t` &`x`)
Search a decreasing vector for the interval containing `x0`.
- `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 find_interval_uncons` (const double `x0`, `size_t` `n`, const `vec_t` &`x`)
Search an increasing or decreasing vector for the interval containing `x0` including the open interval at the end.
- `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 for the interval containing `x0`.
- `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 for the interval containing `x0`.

Protected Attributes

- `size_t cache`
Storage for the most recent index.

8.348.2 Member Function Documentation

8.348.2.1 `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 for the interval containing `x0`.

This function performs a binary search of between `x[lo]` and `x[hi]` (inclusive). It returns

- `lo` if `x0 > x[lo]`
- `i` if `x[i] >= x0 > x[i+1]` for `lo <= i < hi`
- `hi-1` if `x0 <= x[hi]`

The cache is not used for this function.

Note:

The value of `hi` for an `n` element array is typically `n-1`, e.g. For `double x[10]` one would use `bsearch_dec(1.0, x, 0, 9)`.

Definition at line 272 of file `search_vec.h`.

8.348.2.2 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 for the interval containing x0.

This function performs a binary search of between x[lo] and x[hi] (inclusive). It returns

- lo if $x_0 < x[lo]$
- i if $x[i] \leq x_0 < x[i+1]$ for $lo \leq i < hi$
- hi-1 if $x_0 \geq x[hi]$

This function operates in the same way as `gsl_interp_bsearch()`.

The cache is not used for this function.

Note:

The value of hi for an n element array is typically n-1, e.g. For double x[10] one would use `bsearch_inc(1.0, x, 0, 9)`.

Definition at line 241 of file search_vec.h.

8.348.2.3 size_t find_interval_inc (const double x0, size_t n, const vec_t & x) [inline]

Search an increasing vector for the interval containing x0.

This function operates in the same way as `gsl_interp_accel_find()`, except that it does not record cache hits and misses.

Definition at line 102 of file search_vec.h.

8.348.2.4 size_t find_interval_uncons (const double x0, size_t n, const vec_t & x) [inline]

Search an increasing or decreasing vector for the interval containing x0 including the open interval at the end.

This returns the index i for which $x[i] \leq x_0 < x[i+1]$.

This function operates just as `find_interval()`, except that in the case of increasing arrays, it will return n-1 if x0 is greater than the last element in the array. 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.

Definition at line 201 of file search_vec.h.

8.348.2.5 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 x0 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 you have a non-monotonic `ovector` or `uvector` object, consider using `ovector_view_tlate::lookup()` or `uvector_view_tlate::lookup()` instead of this function.

Generally, if there are two adjacent entries with the same value, this function will return the entry with the smaller index. (This is the same as the convention in `ovector_view_tlate::lookup()` and `uvector_view_tlate::lookup()`).

Idea for future

This is not as efficient as it could be, as it just uses the `find_interval` functions and then adjusts the answer at the end if necessary.

Definition at line 158 of file `search_vec.h`.

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

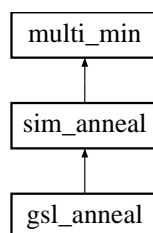
- `search_vec.h`

8.349 `sim_anneal` Class Template Reference

Simulated annealing base.

```
#include <sim_anneal.h>
```

Inheritance diagram for `sim_anneal::`



8.349.1 Detailed Description

template<class param_t, class func_t = multi_func_t, class vec_t = ovector_base, 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 `multi_min::ntrial` which defaults to 100.

Definition at line 57 of file `sim_anneal.h`.

Public Member Functions

- virtual int `mmin` (size_t nvar, vec_t &x, double &fmin, param_t &pa, func_t &func)=0
Calculate the minimum fmin of func w.r.t the array x of size nvar.
- 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.

8.349.2 Member Function Documentation

8.349.2.1 `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 90 of file `sim_anneal.h`.

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

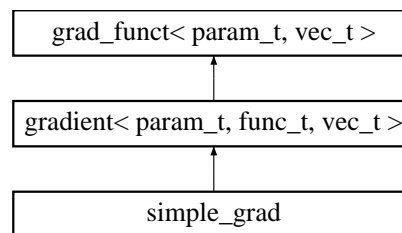
- `sim_anneal.h`

8.350 simple_grad Class Template Reference

Simple automatic computation of [gradient](#) by finite differencing.

```
#include <multi_min.h>
```

Inheritance diagram for `simple_grad`:



8.350.1 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 239 of file `multi_min.h`.

Public Member Functions

- `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} .

Data Fields

- `double epsrel`
The relative stepsize for finite-differencing (default 10^{-6}).
- `double epsmin`
The minimum stepsize (default 10^{-15}).

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

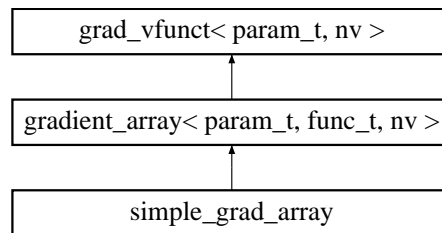
- `multi_min.h`

8.351 simple_grad_array Class Template Reference

Simple automatic computation of [gradient](#) by finite differencing with arrays.

```
#include <multi_min.h>
```

Inheritance diagram for simple_grad_array::



8.351.1 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 475 of file multi_min.h.

Public Member Functions

- virtual int [operator\(\)](#) (size_t nvar, double x[nv], double g[nv], param_t &pa)
Compute the [gradient](#) \mathbf{g} at the point \mathbf{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:

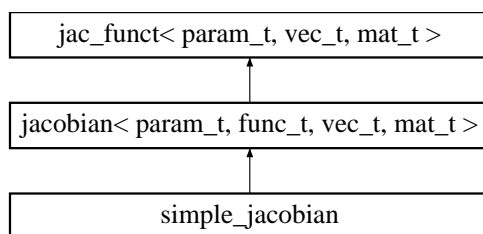
- multi_min.h

8.352 simple_jacobian Class Template Reference

Simple automatic Jacobian.

```
#include <jacobian.h>
```

Inheritance diagram for simple_jacobian::



8.352.1 Detailed Description

`template<class param_t, class func_t = mm_func<param_t>, class vec_t = ovector_base, class mat_t = omatrix_base, 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 class computes a numerical Jacobian by finite differencing. The stepsize is chosen to be $h_j = \text{epsrel } x_j$ or $h_j = \text{epsmin}$ if $\text{epsrel } x_j < \text{epsmin}$.

This is nearly equivalent to the GSL method for computing Jacobians as in `multiroots/fdjac.c`. To obtain the GSL behavior, set `epsrel` to `GSL_SQRT_DBL_EPSILON` and set `epsmin` to zero. The `gsl_mroot_hybrids` class sets `epsrel` to `GSL_SQRT_DBL_EPSILON` in its constructor, but does not set `epsmin` to zero.

This class does not separately check the vector and matrix sizes to ensure they are commensurate.

Idea for future

GSL-1.10 updated `fdjac.c` and this update could be implemented below.

Definition at line 470 of file `jacobian.h`.

Public Member Functions

- 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.
- bool `err_nonconv`
If true, call the error handler if the routine does not "converge".

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

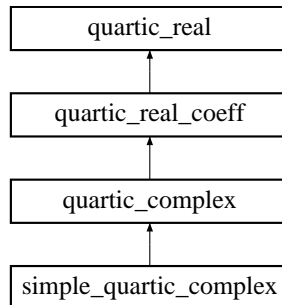
- `jacobian.h`

8.353 simple_quartic_complex Class Reference

Solve a quartic with complex coefficients and complex roots.

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

Inheritance diagram for simple_quartic_complex::



8.353.1 Detailed Description

Solve a quartic with complex coefficients and complex roots.

Definition at line 716 of file poly.h.

Public Member Functions

- 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)
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 $x = x_4$.
- const char * [type](#) ()
Return a string denoting the type ("simple_quartic_complex").

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

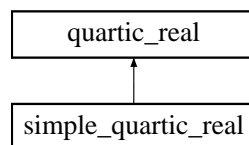
- [poly.h](#)

8.354 simple_quartic_real Class Reference

Solve a quartic with real coefficients and real roots.

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

Inheritance diagram for simple_quartic_real::



8.354.1 Detailed Description

Solve a quartic with real coefficients and real roots.

Definition at line 689 of file poly.h.

Public Member Functions

- 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 ("simple_quartic_real").

Data Fields

- double [cube_root_tol](#)
A tolerance for determining the proper cube [root](#) (default 10^{-6}).

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

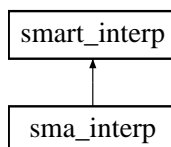
- [poly.h](#)

8.355 sma_interp Class Template Reference

A specialization of [smart_interp](#) for C-style double arrays.

```
#include <smart_interp.h>
```

Inheritance diagram for sma_interp::



8.355.1 Detailed Description

```
template<size_t n> class sma_interp< n >
```

A specialization of [smart_interp](#) for C-style double arrays.

Definition at line 1147 of file smart_interp.h.

Public Member Functions

- [sma_interp](#) (bim1_t &it1, bim2_t &it2, bim3_t &it3, bim4_t &it4)
Create an interpolation object with user-specified interpolation types.
- [sma_interp](#) ()
Create an interpolation object with the default interpolation types.

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

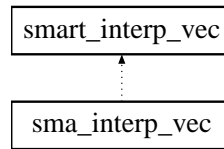
- [smart_interp.h](#)

8.356 sma_interp_vec Class Template Reference

A specialization of [smart_interp_vec](#) for C-style double arrays.

```
#include <smart_interp.h>
```

Inheritance diagram for `sma_interp_vec`:



8.356.1 Detailed Description

```
template<class arr_t> class sma_interp_vec< arr_t >
```

A specialization of [smart_interp_vec](#) for C-style double arrays.

Definition at line 1164 of file `smart_interp.h`.

Public Member Functions

- [sma_interp_vec](#) (bim1_t &it, bim2_t &it2, size_t n, const arr_t &x, const arr_t &y)
Create an interpolation object with user-specified interpolation types.
- [sma_interp_vec](#) (size_t n, const arr_t &x, const arr_t &y)
Create an interpolation object with the default interpolation types.

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

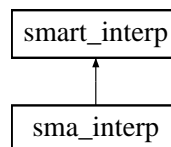
- `smart_interp.h`

8.357 smart_interp Class Template Reference

Smart interpolation class.

```
#include <smart_interp.h>
```

Inheritance diagram for `smart_interp`:



8.357.1 Detailed Description

```
template<class vec_t = ovector_const_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.

Note:

This attempts to determine if the array is increasing or decreasing by examining the relative size of the first and last elements.

Idea for future

Change the determination of the array is increasing to be a bit more intelligent.

Definition at line 53 of file smart_interp.h.

Public Member Functions

- [smart_interp](#) ([base_interp_mgr](#)< [vec_t](#) > &im1, [base_interp_mgr](#)< [rvec_t](#) > &im2, [base_interp_mgr](#)< [svec_t](#) > &im3, [base_interp_mgr](#)< [srvec_t](#) > &im4)
Create a new interpolation object with the specified interpolation managers.
- [smart_interp](#) ()
Create an interpolation object with the default cubic spline interpolation managers.
- 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 managers

- [def_interp_mgr](#)< [vec_t](#), [cspline_interp](#) > **dim1**
- [def_interp_mgr](#)< [rvec_t](#), [cspline_interp](#) > **dim2**
- [def_interp_mgr](#)< [svec_t](#), [cspline_interp](#) > **dim3**
- [def_interp_mgr](#)< [srvec_t](#), [cspline_interp](#) > **dim4**

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

- bool **sxalloc**
- const [svec_t](#) * **sx**
- const [svec_t](#) * **sy**
- bool **srxalloc**
- 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`

Pointers to interpolation managers

- `base_interp_mgr< vec_t > * bim1`
- `base_interp_mgr< rvec_t > * bim2`
- `base_interp_mgr< svec_t > * bim3`
- `base_interp_mgr< srvec_t > * bim4`

8.357.2 Member Function Documentation

8.357.2.1 `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.

This function tries to find the largest monotonic region enclosing both a and b in the vector x. If it succeeds, it returns `gsl_success`, and if it fails, it returns `gsl_efaied`. It does not call the error handler.

Todo

The error handling is a bit off here, as it can return a non-zero value even with there is no real "error". We should just make a new bool reference paramter.

Definition at line 623 of file smart_interp.h.

8.357.2.2 `virtual double interp (const double x0, size_t n, const vec_t & x, const vec_t & y)` [*inline, virtual*]

Give the value of the function $y(x = x_0)$.

Todo

After calling find_subset, I think we might need to double check that nn is larger than the minimum interpolation size.

Definition at line 167 of file smart_interp.h.

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

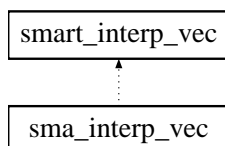
- smart_interp.h

8.358 smart_interp_vec Class Template Reference

Smart interpolation class with pre-specified vectors.

```
#include <smart_interp.h>
```

Inheritance diagram for smart_interp_vec::



8.358.1 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.

Idea for future

Properly implement handling of non-monotonic regions in the derivative functions as well as the interpolation function.

Definition at line 747 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_mgr< vec_t > &it1, base_interp_mgr< svec_t > &it2, size_t n, const vec_t &x, const vec_t &y)
Create with base interpolation objects `it` and `rit`.
- 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

- [def_interp_mgr< vec_t, cspline_interp > dim1](#)
Default interpolation manager.
- [def_interp_mgr< svec_t, cspline_interp > dim2](#)
Default interpolation manager.

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

- bool [sxalloc](#)
If true, then `sx` and `sy` have been allocated.
- 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.
- [base_interp_mgr](#)< vec_t > * [bim1](#)
Pointer to base interpolation manager.
- [base_interp_mgr](#)< svec_t > * [bim2](#)
Pointer to base interpolation manager.
- [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.

8.358.2 Member Function Documentation

8.358.2.1 `int find_inc_subset (const double $x0$, size_t sz , const vec_t & x , const vec_t & y , size_t & nsz)` [`inline`, `protected`]

Try to find the largest monotonic and finite region around the desired location.

This function looks through the vector x near the element closest to $x0$ to find the largest possible monotonic region. If it succeeds, it returns [gsl_success](#), and if it fails, it returns [gsl_efailed](#). It does not call the error handler.

Definition at line 1044 of file `smart_interp.h`.

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

- `smart_interp.h`

8.359 string_comp Struct Reference

Simple string comparison.

```
#include <misc.h>
```

8.359.1 Detailed Description

Simple string comparison.

This struct is used internally by `O2scl` for the STL routines which require a way to compare strings in the class [table](#) and in the I/O classes.

Definition at line 168 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:

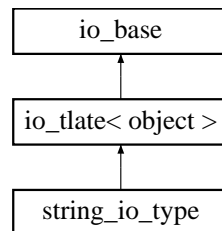
- [misc.h](#)

8.360 string_io_type Class Reference

I/O object for string variables.

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

Inheritance diagram for string_io_type::



8.360.1 Detailed Description

I/O object for string variables.

This class is experimental.

Definition at line 2364 of file collection.h.

Public Member Functions

- [string_io_type](#) (const char *t)
Desc.

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

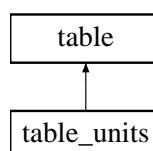
- [collection.h](#)

8.361 table Class Reference

Data table class.

```
#include <table.h>
```

Inheritance diagram for table::



8.361.1 Detailed Description

Data table class.

Summary

A class to contain and manipulate several equally-sized columns of data. The purpose of this class is to provide a structure which allows one to refer to the columns using a name represented by a string. Thus for a [table](#) object named `t` with 3 columns (named "colx", "coly" and "colz") and three rows, one could do the following:

```
// Set the 1st row of column "colx" to 1.0
t.set("colx",0,1.0);
// Set the 2nd row of column "colz" to 2.0
t.set("colz",1,2.0);
// Set the 3rd row of column "coly" to 4.0
t.set("coly",2,4.0);
// This will print out 2.0
cout << t.get("colz",1) << endl;
```

Note that the rows are numbered starting with 0 instead of starting with 1. To output all the rows of entire column, one can use

```
for(size_t i=0;i<t.get_nlines();i++) {
    cout << i << " " << t.get("colx",i) << endl;
}
```

To output all the columns of an entire row (in the following example it is the second row), labeled by their column name, one can use:

```
for(size_t i=0;i<t.get_ncolumns();i++) {
    cout << t.get_column_name(i) << " ";
}
cout << endl;
for(size_t i=0;i<t.get_ncolumns();i++) {
    cout << t.get(i,1) << " ";
}
cout << endl;
```

Methods are provided for interpolating columns, sorting columns, finding data points, and several other manipulations of the data.

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 $O(R \times C)$. 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_vec](#). 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 ([sort_column\(\)](#)), or the entire [table](#) can be sorted by one column ([sort_table\(\)](#)).

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 `O2scl_ext` extension to this class named `table_fp`, however requires that the column name contain only letters, numbers, and underscores and also that the column name not begin with a digit.

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()` 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.

Because of the structure, this class is not suitable for the matrix manipulation. The classes `omatrix` and `umatrix` are better used for that purpose.

Thread-safety

Generally, the member functions are thread-safe in the sense that one would expect: it is always permitted to have many threads accessing many tables, but care must be taken if many threads are accessing only one `table`. Simple `get()` and `set()` functions are thread-safe (unless a `set()` function forces memory rearrangement), while insertion and deletion operations are not. Only the `const` version of the interpolation routines are 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.

There is an example for the usage of this class given in `examples/ex_table.cpp`.

Idea for future

Rewrite the `table::create_array()` and `table::insert_data()` functions for generic vector type

Idea for future

Be more restrictive about allowable column names?

Idea for future

Re-implement user-owned columns, and handle automatic resizing appropriately.

Idea for future

Return the empty column in the `operator[]` functions as is done for the `get_column()` functions.

Idea for future

A "delete rows" method to delete a range of several rows

Idea for future

The present structure, `std::map<std::string, col, string_comp> atree` and `std::vector<aiter> alist`; could be replaced with `std::vector<col> list` and `std::map<std::string, int> tree` where the map just stores the index of the the column in the list

Definition at line 207 of file `table.h`.

Data Structures

- struct `col_s`
Column structure for `table` [protected].
- struct `sortd_s`
A structure for sorting in `table` [protected].

Public Member Functions

- `table` (int `cmaxlines`=0)
Create a new `table` with space for `nlines` ≤ `cmaxlines`.
- `table` (const `table` &`t`)
Copy constructor.
- `table` & `operator=` (const `table` &`t`)
Copy constructor.
- virtual const char * `type` ()
Return the type, "table".

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`) const
Get value from row `row` of column number `icol` - $O(1)$.
- size_t `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, increasing the size more aggressively.
- int `get_maxlines` ()
Return the maximum number of lines.
- `ovector_base` & `get_column` (std::string `col`)
Returns a reference to the column named `col` - $O(\log(C))$.
- const `ovector_base` & `get_column` (std::string `col`) const
Returns a reference to the column named `col` - $O(\log(C))$.
- const `ovector_base` & `operator[]` (size_t `icol`) const
Returns the column of index `icol` - $O(1)$ (const version).
- `ovector_base` & `operator[]` (size_t `icol`)
Returns the column of index `icol` - $O(1)$.
- const `ovector_base` & `operator[]` (std::string `scol`) const
Returns the column named `scol` - $O(\log(C))$ (const version).
- `ovector_base` & `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 copy of the row with value `val` in column `col` - $O(R*C)$.*
- int `get_row` (size_t `irow`, `ovector` &`row`) const
Returns 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` - $O(\log(C))$.
- `bool is_column (std::string scol)`
Return true if `scol` is a column in the current `table`.
- `int lookup_column (std::string name, int &ix) const`
Find the index for column named `name` - $O(\log(C))$.
- `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` - $O(\log(C)*R)$.*
- `const gsl_vector * get_gsl_vector (std::string name) const`
Get a `gsl_vector` from column `name` - $O(\log(C))$.
- `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` - $O(\log(C)*R)$.*

Interpolation, differentiation, and integration, max, and min

- `int set_interp (base_interp_mgr< ovector_const_view > &bi1, base_interp_mgr< ovector_const_subvector > &bi2)`
Set the base interpolation objects.
- `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`.
- `double interp_const (size_t ix, double x0, size_t iy) const`
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_const** (std::string sx, double x0, std::string sy) const
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 $ix=x0$.
- double **deriv_const** (size_t ix, double x0, size_t iy) const
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_const** (std::string sx, double x0, std::string sy) const
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 **deriv2_const** (size_t ix, double x0, size_t iy) const
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_const** (std::string sx, double x1, double x2, std::string sy) const
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 $ix=x2$.
- double **integ_const** (size_t ix, double x1, double x2, size_t iy) const
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) const
Make a subtable.

Add space

- int **inc_maxlines** (size_t llines)
Manually increase the maximum number of lines.

Delete methods

- virtual 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 entry closest to the value val in column scol - $O(R*C)$.*
- int **delete_row** (size_t irow)
*Delete the row of index irow - $O(R*C)$.*

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

- virtual int [summary](#) (std::ostream *out, int ncol=79) const
Output a summary of the information stored.

Constant manipulation

- virtual int [add_constant](#) (std::string name, double val)
Add a constant, or if the constant already exists, change its value.
- virtual int [set_constant](#) (std::string name, double val, bool err_on_notfound=true)
Add a constant.
- virtual double [get_constant](#) (std::string name) const
Get a constant.
- virtual size_t [get_nconsts](#) () const
Get the number of constants.
- virtual int [get_constant](#) (size_t ix, std::string &name, double &val) const
Get a constant by index.
- virtual int [remove_constant](#) (std::string name)
Remove a constant.

Miscellaneous methods

- int [read_generic](#) (std::istream &fin)
Clear the current [table](#) and read from a generic data file.
- int [check_synchro](#) () const
Return 0 if the tree and list are properly synchronized.
- bool [get_owner](#) (std::string name) const
Get ownership - $O(\log(C))$.

Data Fields

- [def_interp_mgr](#)< [ovector_const_view](#), [cspline_interp](#) > dim1
Default interpolation manager.
- [def_interp_mgr](#)< [ovector_const_subvector](#), [cspline_interp](#) > dim2
Default interpolation manager.
- bool [intp_set](#)
True if the interpolation type has been set.

Protected Types

- typedef struct [table::col_s](#) col
Column structure for [table](#) [protected].
- typedef struct [table::sortd_s](#) sortd
A structure for sorting in [table](#) [protected].

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 - $O(C)$.
- int [make_fp_varname](#) (std::string &s)
Ensure a variable name does not match a function or contain non-alphanumeric characters.
- int [make_unique_name](#) (std::string &col, std::vector< std::string > &cnames)
Make sure a name is unique.

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

- std::map< std::string, double > [constants](#)
The list of constants.
- [ovector empty_col](#)
An empty vector for [get_column\(\)](#).

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_mgr](#)< [ovector_const_view](#) > * [bim1](#)
A pointer to the interpolation manager.
- [base_interp_mgr](#)< [ovector_const_subvector](#) > * [bim2](#)
A pointer to the subvector interpolation manager.
- [search_vec](#)< [ovector](#) > [se](#)
The vector-searching object.
- std::string [intp_colx](#)
The last x-column interpolated.
- std::string [intp_coly](#)
The last y-column interpolated.

8.361.2 Member Function Documentation

8.361.2.1 `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 objects from the `source` [table](#) will be used. 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.

8.361.2.2 `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 792 of file `table.h`.

8.361.2.3 `virtual int delete_column (std::string scol) [virtual]`

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.

Reimplemented in [table_units](#).

8.361.2.4 `double deriv (size_t ix, double x0, size_t iy)`

The first derivative of the function `iy(ix)` at `ix=x0`.

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

8.361.2.5 `double deriv (std::string sx, double x0, std::string sy)`

The first derivative of the function `sy(sx)` at `sx=x0`.

O(log(C)*log(R)) but can be as bad as O(log(C)*R) if the relevant columns are not well ordered.

8.361.2.6 `double deriv2 (size_t ix, double x0, size_t iy)`

The second derivative of the function `iy(ix)` at `ix=x0`.

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

8.361.2.7 `double deriv2 (std::string sx, double x0, std::string sy)`

The second derivative of the function `sy(sx)` at `sx=x0`.

O(log(C)*log(R)) but can be as bad as O(log(C)*R) if the relevant columns are not well ordered.

8.361.2.8 `double deriv2_const (size_t ix, double x0, size_t iy) const`

The second derivative of the function `iy(ix)` at `ix=x0`.

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

8.361.2.9 double deriv2_const (std::string sx, double x0, std::string sy) const

The second derivative of the function sy(sx) at sx=x0.

$O(\log(C)*\log(R))$ but can be as bad as $O(\log(C)*R)$ if the relevant columns are not well ordered.

8.361.2.10 double deriv_const (size_t ix, double x0, size_t iy) const

The first derivative of the function iy(ix) at ix=x0.

$O(\log(R))$ but can be as bad as $O(R)$ if the relevant columns are not well ordered.

8.361.2.11 double deriv_const (std::string sx, double x0, std::string sy) const

The first derivative of the function sy(sx) at sx=x0.

$O(\log(C)*\log(R))$ but can be as bad as $O(\log(C)*R)$ if the relevant columns are not well ordered.

8.361.2.12 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.

8.361.2.13 int integ (std::string x, std::string y, std::string ynew)

The integral of the function iy(ix).

$O(\log(R))$ but can be as bad as $O(R)$ if the relevant columns are not well ordered.

8.361.2.14 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.

$O(\log(R))$ but can be as bad as $O(R)$ if the relevant columns are not well ordered.

8.361.2.15 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.

$O(\log(C)*\log(R))$ but can be as bad as $O(\log(C)*R)$ if the relevant columns are not well ordered.

8.361.2.16 double integ_const (size_t ix, double x1, double x2, size_t iy) const

The integral of the function iy(ix) from ix=x1 to ix=x2.

$O(\log(R))$ but can be as bad as $O(R)$ if the relevant columns are not well ordered.

8.361.2.17 double integ_const (std::string sx, double x1, double x2, std::string sy) const

The integral of the function sy(sx) from sx=x1 to sx=x2.

$O(\log(C)*\log(R))$ but can be as bad as $O(\log(C)*R)$ if the relevant columns are not well ordered.

8.361.2.18 double interp (size_t ix, double x0, size_t iy)

Interpolate x0 from ix into iy.

$O(\log(R))$ but can be as bad as $O(R)$ if the relevant columns are not well ordered.

8.361.2.19 double interp (std::string *sx*, double *x0*, std::string *sy*)

Interpolate *x0* from *sx* into *sy*.

$O(\log(C) \cdot \log(R))$ but can be as bad as $O(\log(C) \cdot R)$ if the relevant columns are not well ordered.

8.361.2.20 double interp_const (size_t *ix*, double *x0*, size_t *iy*) const

Interpolate *x0* from *ix* into *iy*.

$O(\log(R))$ but can be as bad as $O(R)$ if the relevant columns are not well ordered.

8.361.2.21 double interp_const (std::string *sx*, double *x0*, std::string *sy*) const

Interpolate *x0* from *sx* into *sy*.

$O(\log(C) \cdot \log(R))$ but can be as bad as $O(\log(C) \cdot R)$ if the relevant columns are not well ordered.

8.361.2.22 bool is_column (std::string *scol*)

Return true if *scol* is a column in the current [table](#).

This function does not call the error handler if the column is not found, but just silently returns false.

8.361.2.23 int lookup_column (std::string *name*, int &*ix*) const

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_enotfound](#).

8.361.2.24 int new_row (size_t *n*)

Insert a row before row *n*.

Acceptable values for *n* are between 0 and [get_nlines\(\)](#) inclusive, with the maximum value denoting the addition of a row after the last row presently in the [table](#).

8.361.2.25 ovector_base& 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 340 of file [table.h](#).

8.361.2.26 const ovector_base& 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 326 of file [table.h](#).

8.361.2.27 ovector_base& 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 313 of file table.h.

8.361.2.28 `const ovector_base& 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 299 of file table.h.

8.361.2.29 `size_t ordered_lookup (std::string col, double val)`

Look for a value in an ordered column.

$O(\log(C) * \log(R))$

This uses the function `search_vec::ordered_lookup()`, which offers caching and assumes the vector is monotonic. If you don't have monotonic data, you can still use the `table::lookup()` function, which is more general.

8.361.2.30 `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.

8.361.2.31 `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.

8.361.2.32 `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`.

8.361.2.33 `int set_nlines_auto (size_t il)`

Set the number of lines, increasing the size more aggressively.

This function is like `set_nlines()`, but doubles the maximum column size if an increase in the maximum size is required instead of simply making enough room for the current number of lines. This function is used internally by `set()` to ensure that the cost of setting lines in sequence is linear and not quadratic.

8.361.2.34 `table* subtable (std::string list, size_t top, size_t bottom) const`

Make a subtable.

Uses the columns specified in `list` from the row `top` to the row of index `bottom` to generate a new `table` which is a copy of part of the original.

8.361.2.35 virtual int summary (std::ostream * out, int ncol = 79) const [virtual]

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.

Reimplemented in [table_units](#).

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

- [table.h](#)

8.362 table3d Class Reference

A data structure containing many slices of two-dimensional data points defined on a grid.

```
#include <table3d.h>
```

8.362.1 Detailed Description

A data structure containing many slices of two-dimensional data points defined on a grid.

Idea for future

Improve interpolation and derivative caching

Idea for future

Make a 'const' version of the interpolation functions

Definition at line 53 of file [table3d.h](#).

Public Member Functions

- [table3d](#) ()
Create a new 3D table .
- virtual const char * [type](#) ()
Return the type, "table3d".

Initialization

- template<class vec2_t>
int [set_xy](#) (std::string x_name, size_t nx, const vec2_t &x, std::string y_name, size_t ny, const vec2_t &y)
Initialize the x-y grid.
- int [set_size](#) (size_t nx, size_t ny)
Initialize table size.

On-grid get and set methods

- int [set](#) (size_t ix, size_t iy, std::string name, double val)
Set element in slice name at location ix, iy to value val.
- int [set](#) (size_t ix, size_t iy, size_t z, double val)
Set element in slice of index z at location ix, iy to value val.
- double & [get](#) (size_t ix, size_t iy, std::string name)
Get element in slice name at location ix, iy.
- const double & [get](#) (size_t ix, size_t iy, std::string name) const
Get element in slice name at location ix, iy (const version).
- double & [get](#) (size_t ix, size_t iy, size_t z)

Get element in slice of index z at location ix, iy.

- const double & [get](#) (size_t ix, size_t iy, size_t z) const
Get element in slice of index z at location ix, iy (const version).

Off-grid get and set methods

These methods return the value of a slice on the grid point nearest to a user-specified location. For interpolation into a point off the grid, use [table3d::interp\(\)](#).

- int [set_val](#) (double x, double y, std::string name, double val)
Set element in slice name at the nearest location to x, y to value val.
- int [set_val](#) (double x, double y, size_t z, double val)
Set element in slice of index z at the nearest location to x, y to value val.
- double & [get_val](#) (double x, double y, std::string name)
Get element in slice name at location closest to x, y, and also return the corresponding values of x and y.
- const double & [get_val](#) (double x, double y, std::string name) const
Get element in slice name at location closest to x, y, and also return the corresponding values of x and y.
- double & [get_val](#) (double x, double y, size_t z)
Get element in slice of index z at location closest to x, y, and also return the corresponding values of x and y.
- const double & [get_val](#) (double x, double y, size_t z) const
Get element in slice of index z at location closest to x, y, and also return the corresponding values of x and y.
- template<class vec_t >
int [set_slices](#) (double x, double y, size_t nv, vec_t &vals)
Set elements in the first nv slices at the nearest location to x, y to value val.
- template<class vec_t, class alloc_t >
int [get_point](#) (double x, double y, vec_t &v)
Get all the slice data from a point.

Off-grid get and set methods returning nearest point

- int [set_val_ret](#) (double &x, double &y, std::string name, double val)
Set element in slice name at the nearest location to x, y to value val.
- int [set_val_ret](#) (double &x, double &y, size_t z, double val)
Set element in slice of index z at the nearest location to x, y to value val.
- double & [get_val_ret](#) (double &x, double &y, std::string name)
Get element in slice name at location closest to x, y, and also return the corresponding values of x and y.
- const double & [get_val_ret](#) (double &x, double &y, std::string name) const
Get element in slice name at location closest to x, y, and also return the corresponding values of x and y.
- double & [get_val_ret](#) (double &x, double &y, size_t z)
Get element in slice of index z at location closest to x, y, and also return the corresponding values of x and y.
- const double & [get_val_ret](#) (double &x, double &y, size_t z) const
Get element in slice of index z at location closest to x, y, and also return the corresponding values of x and y.
- template<class vec_t >
int [set_slices_ret](#) (double &x, double &y, size_t nv, vec_t &vals)
Set elements in the first nv slices at the nearest location to x, y to value val.

Grid information get and set methods

- int [set_grid_x](#) (size_t ix, double val)
Set x grid point at index ix.
- int [set_grid_y](#) (size_t iy, double val)
Set y grid point at index iy.
- double [get_grid_x](#) (size_t ix)
Get x grid point at index ix.
- double [get_grid_y](#) (size_t iy)
Get y grid point at index iy.
- std::string [get_x_name](#) ()
Get the name of the x grid variable.
- std::string [get_y_name](#) ()
Get the name of the y grid variable.
- int [set_x_name](#) (std::string name)

- *Set the name of the x grid variable.*
- int `set_y_name` (std::string name)
Set the name of the y grid variable.
- const `ovector` & `get_x_data` () const
Get a const reference to the full x grid.
- const `ovector` & `get_y_data` () const
Get a const reference to the full y grid.

Size get methods

- int `get_size` (size_t &nx, size_t &ny) const
Get the size of each slice.
- int `get_nx` () const
Get the x size.
- int `get_ny` () const
Get the y size.
- int `get_nslices` () const
Get the number of slices.

Slice manipulation

- int `line_of_names` (std::string names)
Create a set of new slices specified in the string names.
- std::string `get_slice_name` (size_t col) const
Returns the name of slice with index col.
- int `new_slice` (std::string name)
Add a new column owned by the table .
- int `lookup_slice` (std::string name, size_t &ix) const
Find the index for column named name.
- bool `is_slice` (std::string name, int &ix)
Return true if slice is already present.
- int `rename_slice` (std::string olds, std::string news)
Rename slice named olds to news.
- int `copy_slice` (std::string src, std::string dest)
Make a new column named dest equal to src.
- int `init_slice` (std::string scol, double val)
Initialize all values of slice named scol to val.
- const `omatrix` & `get_slice` (std::string scol) const
Return a constant reference to a slice.
- const `omatrix` & `get_slice` (size_t iz) const
Return a constant reference to a slice.
- const std::vector< `omatrix` > & `get_data` () const
Return a constant reference to all the slice data.

Lookup and search methods

- int `lookup_x` (double val, size_t &ix) const
Look for a value in the x grid.
- int `lookup_y` (double val, size_t &iy) const
Look for a value in the y grid.
- int `lookup` (double val, std::string slice, size_t &ix, size_t &iy) const
Look for a value in the grid.

Interpolation, differentiation, and integration

- int `set_interp` (base_interp_mgr< `ovector_const_view` > &b1, base_interp_mgr< `ovector_const_subvector` > &b2)
Specify the base interpolation objects to use.
- double `interp` (double x, double y, std::string name)
Interpolate x and y in slice named name.
- double `deriv_x` (double x, double y, std::string name)
Interpolate the derivative of the data with respect to the x grid at point x and y in slice named name.

- double `deriv_y` (double x, double y, std::string name)
Interpolate the derivative of the data with respect to the y grid at point x and y in slice named name.
- double `deriv_xy` (double x, double y, std::string name)
Interpolate the mixed second derivative of the data at point x and y in slice named name.
- double `integ_x` (double x1, double x2, double y, std::string name)
Interpolate the integral of the data respect to the x grid.
- double `integ_y` (double x, double y1, double y2, std::string name)
Interpolate the integral of the data respect to the y grid.

Extract 2-dimensional tables

- int `extract_x` (double x, `table` &t)
Extract a `table` at a fixed x grid point.
- int `extract_y` (double y, `table` &t)
Extract a `table` at a fixed y grid point.

Clear methods

- int `zero_table` ()
Zero the data entries but keep the column names and nlines fixed.
- void `clear_table` ()
Clear the `table` and the slice names.
- void `clear_data` ()
Remove all of the data by setting the number of lines to zero.

Summary method

- int `summary` (std::ostream *out, int ncol=79) const
Output a summary of the information stored.

Manipulating constants

- virtual int `add_constant` (std::string name, double val)
Add a constant.
- virtual int `remove_constant` (std::string name)
Remove a constant.
- virtual int `set_constant` (std::string name, double val)
Add a constant.
- virtual double `get_constant` (std::string name)
Get a constant.

Data Fields

Default interpolation objects

- `def_interp_mgr`< `ovector_const_view`, `cspline_interp` > `dim1`
- `def_interp_mgr`< `ovector_const_subvector`, `cspline_interp` > `dim2`

Protected Types

Iterator types

- typedef std::map< std::string, size_t, `string_comp` >::iterator `map_iter`
- typedef std::map< std::string, size_t, `string_comp` >::const_iterator `map_const_iter`

Protected Member Functions

Tree iterator boundaries

- `map_iter begin ()`
Return the beginning of the slice tree.
- `map_iter end ()`
Return the end of the slice tree.
- `map_const_iter const_begin () const`
Return the beginning of the slice tree.
- `map_const_iter const_end () const`
Return the end of the slice tree.

Protected Attributes

Interpolation data

- `base_interp_mgr< ovector_const_view > * bimp1`
The base interpolation object.
- `base_interp_mgr< ovector_const_subvector > * bimp2`
The subvector base interpolation object.
- `sm_interp_vec ** si`
The array of sm_interp_vec pointers.
- `omatrix_col ** aci`
Matrices for interpolation.

Data storage

- `std::map< std::string, double > constants`
The list of constants.
- `size_t numx`
The size of the x grid.
- `size_t numy`
The size of the y grid.
- `std::map< std::string, size_t, string_comp > tree`
A tree connecting column names to list indexes.
- `std::string xname`
The name for the x grid.
- `std::string yname`
The name for the y grid.
- `std::vector< omatrix > list`
The pointers to the matrices.
- `ovector xval`
The x grid.
- `ovector yval`
The y grid.
- `bool xy_set`
True if the grid has been set.
- `bool size_set`
True if the size of the grid has been set.
- `bool has_slice`
True if the table has at least one slice.

8.362.2 Member Function Documentation

8.362.2.1 void clear_data ()

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.

8.362.2.2 `int init_slice (std::string scol, double val)`

Initialize all values of slice named `scol` to `val`.

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.

8.362.2.3 `int rename_slice (std::string olds, std::string news)`

Rename slice named `olds` to `news`.

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.

8.362.2.4 `int set_size (size_t nx, size_t ny)`

Initialize table size.

This function will not allow you to resize the table if it already has data or if the size has already been set with the `set_xy()` function, unless you clear the data with `clear_data()` or the table with `clear_table()` first.

**8.362.2.5 `int set_xy (std::string x_name, size_t nx, const vec2_t & x, std::string y_name, size_t ny, const vec2_t & y)`
 `[inline]`**

Initialize the x-y grid.

This function will not allow you to redefine the grid when there is data in the table if a grid of a different size was already set from a previous call to either `set_xy()` or `set_size()`. However, you may freely redefine the grid after a call to `clear_data()` or `clear_table()`. You may change individual grid points at any time with `set_grid_x()` and `set_grid_y()`.

Definition at line 77 of file `table3d.h`.

8.362.2.6 `int summary (std::ostream * out, int ncol = 79) const`

Output a summary of the information stored.

Outputs the number of constants, the grid information, and a list of the slice names

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

- `table3d.h`

8.363 `table::col_s` Struct Reference

Column structure for `table` [protected].

```
#include <table.h>
```

8.363.1 Detailed Description

Column structure for `table` [protected].

Definition at line 938 of file `table.h`.

Data Fields

- `ovector_base * 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

8.364 **table::sortd_s Struct Reference**

A structure for sorting in [table](#) [protected].

```
#include <table.h>
```

8.364.1 Detailed Description

A structure for sorting in [table](#) [protected].

Definition at line 985 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:

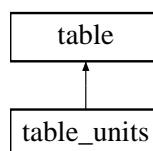
- table.h

8.365 **table_units Class Reference**

Data table class with units.

```
#include <table_units.h>
```

Inheritance diagram for table_units::



8.365.1 Detailed Description

Data table class with units.

Todo

Make [table](#) methods virtual? (not necessary yet since [delete_column\(\)](#) isn't referred to internally)

acol commands which need editing for [table_units](#)

- binary
- html

Definition at line 43 of file `table_units.h`.

Public Member Functions

- [table_units](#) (int cmaxlines=0)
Create a new [table_units](#) with space for nlines<=cmaxlines.
- std::string [get_unit](#) (std::string scol) const
Get the unit for column scol.
- int [remove_unit](#) (std::string scol)
Remove the unit for column scol.
- int [set_unit](#) (std::string scol, std::string unit)
Set the unit for column scol to unit.
- int [convert_to_unit](#) (std::string scol, std::string unit, bool err_on_fail=true)
Convert the units of column scol to unit.
- double [get_conv](#) (std::string old_unit, std::string new_unit)
Get the conversion factor from old_unit to new_unit.
- virtual int [delete_column](#) (std::string scol)
Delete column named scol.
- virtual const char * [type](#) ()
Return the type, "table_units".
- virtual int [summary](#) (std::ostream *out, int ncol=79) const
Output a summary of the information stored.
- int [set_convert](#) ([convert_units](#) &c)
Set the convert units object.
- int [show_units](#) ()
Show the units.

Copy constructors

- **table_units** (const [table_units](#) &t)
- **table_units** (const [table](#) &t)
- [table_units](#) & **operator=** (const [table_units](#) &t)
- [table_units](#) & **operator=** (const [table](#) &t)
Copy constructor.

Data Fields

- [convert_units](#) [def_cu](#)
The default object for unit conversions.

Protected Types

Unit map iterator types

- typedef std::map< std::string, std::string, [string_comp](#) >::iterator **uiter**
- typedef std::map< std::string, std::string, [string_comp](#) >::const_iterator **uciter**

Protected Attributes

- `convert_units * cup`
The pointer to the convert units object.
- `std::map< std::string, std::string, string_comp > utree`
Unit map.

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

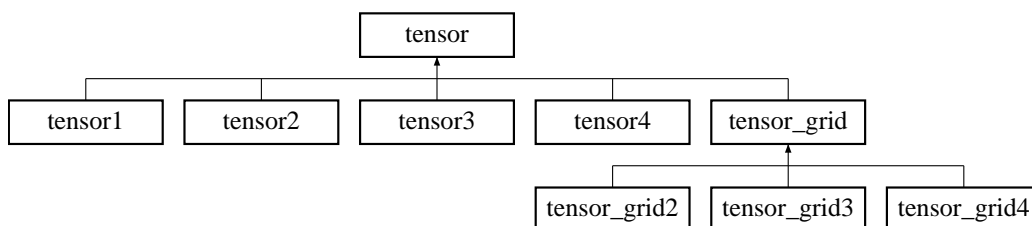
- `table_units.h`

8.366 tensor Class Reference

Tensor class with arbitrary dimensions.

```
#include <tensor.h>
```

Inheritance diagram for tensor::



8.366.1 Detailed Description

Tensor class with arbitrary dimensions.

Todo

More complete testing.

Idea for future

Could implement arithmetic operators + and - and some different products.

Idea for future

Add slicing to get `ovector` or `omatrix` objects

Definition at line 57 of file `tensor.h`.

Public Member Functions

- `tensor ()`
Create an empty `tensor` with zero rank.
- `tensor (size_t rank, size_t *dim)`
Create a `tensor` of rank `rank` with sizes given in `dim`.
- `virtual int set (size_t *index, double val)`
Set the element indexed by `index` to value `val`.
- `virtual double & get (size_t *index)`

- *Get the element indexed by `index`.*
- virtual double const & `get` (size_t *`index`) const
Get the element indexed by `index`.
- `ovector_array_stride vector_slice` (size_t `ix`, size_t *`index`)
Fix all but one index to create a vector.
- `omatrix_array matrix_slice` (size_t *`index`, size_t `ix`)
Fix all but two indices to create a matrix.
- virtual int `get_rank` ()
Return the rank of the `tensor`.
- virtual int `tensor_allocate` (size_t `rank`, size_t *`dim`)
Allocate space for a `tensor` of rank `rank` with sizes given in `dim`.
- virtual int `tensor_free` ()
Free allocated space (also sets rank to zero).
- virtual size_t `get_size` (size_t `i`)
Returns the size of the `i`th index.
- virtual size_t `total_size` ()
Returns the size of the `tensor`.
- size_t `pack_indices` (size_t *`index`)
Pack the indices into a single array index.
- int `unpack_indices` (size_t `ix`, size_t *`index`)
Unpack the single array index into indices.

Protected Attributes

- double * `data`
- size_t * `size`
A rank-sized array of the sizes of each dimension.
- size_t `rk`
Rank.

8.366.2 Constructor & Destructor Documentation

8.366.2.1 `tensor` (size_t `rank`, size_t *`dim`) [inline]

Create a `tensor` of rank `rank` with sizes given in `dim`.

The parameter `dim` must be a pointer to an array of sizes with length `rank`. If the user requests any of the sizes to be zero, this constructor will call the error handler, create an empty `tensor`, and will allocate no memory.

Definition at line 91 of file `tensor.h`.

8.366.3 Member Function Documentation

8.366.3.1 `omatrix_array matrix_slice` (size_t *`index`, size_t `ix`) [inline]

Fix all but two indices to create a matrix.

This fixes all of the indices to the values given in `index` except for the index number `ix` and the last index, and returns the corresponding matrix, whose size is equal to the size of the `tensor` in the two indices which are not fixed.

Definition at line 246 of file `tensor.h`.

8.366.3.2 `virtual int tensor_allocate` (size_t `rank`, size_t *`dim`) [inline, virtual]

Allocate space for a `tensor` of rank `rank` with sizes given in `dim`.

The parameter `dim` must be a pointer to an array of sizes with length `rank`.

If memory was previously allocated, it will be freed before the new allocation and previously specified grid data will be lost.

If the user requests any of the sizes to be zero, this function will call the error handler and will allocate no memory. If memory was previously allocated, the [tensor](#) is left unmodified and no deallocation is performed.

Reimplemented in [tensor_grid](#).

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

8.366.3.3 `ovector_array_stride vector_slice (size_t ix, size_t *index) [inline]`

Fix all but one index to create a vector.

This fixes all of the indices to the values given in `index` except for the index number `ix`, and returns the corresponding vector, whose length is equal to the size of the [tensor](#) in that index. The value `index[ix]` is ignored.

For example, for a rank 3 [tensor](#) allocated with

```
tensor t;
size_t dim[3]={3,4,5};
t.tensor_allocate(3,dim);
```

the following code

```
size_t index[3]={1,0,3};
ovector_view v=t.vector_slice(index,1);
```

Gives a vector `v` of length 4 which refers to the values `t(1,0,3)`, `t(1,1,3)`, `t(1,2,3)`, and `t(1,3,3)`.

Definition at line 226 of file `tensor.h`.

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

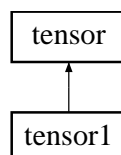
- [tensor.h](#)

8.367 **tensor1 Class Reference**

Rank 1 [tensor](#).

```
#include <tensor.h>
```

Inheritance diagram for `tensor1`:



8.367.1 Detailed Description

Rank 1 [tensor](#).

Definition at line 788 of file `tensor.h`.

Public Member Functions

- [tensor1 \(\)](#)
Create an empty [tensor](#).

- [tensor1](#) (size_t sz)
Create a rank 1 tensor of size sz.
- virtual double & [get](#) (size_t ix)
Get the element indexed by ix.
- virtual const double & [get](#) (size_t ix) const
Get the element indexed by ix.
- virtual int [set](#) (size_t index, double val)
Set the element indexed by index to value val.
- virtual double & [operator\[\]](#) (size_t ix)
Get an element using array-like indexing.
- virtual double & [operator\(\)](#) (size_t ix)
Get an element using operator().

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

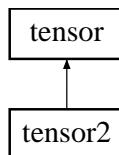
- [tensor.h](#)

8.368 tensor2 Class Reference

Rank 2 [tensor](#).

```
#include <tensor.h>
```

Inheritance diagram for tensor2::



8.368.1 Detailed Description

Rank 2 [tensor](#).

Definition at line 817 of file tensor.h.

Public Member Functions

- [tensor2](#) ()
Create an empty [tensor](#).
- [tensor2](#) (size_t sz, size_t sz2)
Create a rank 2 [tensor](#) of size (sz,sz2).
- virtual double & [get](#) (size_t ix1, size_t ix2)
Get the element indexed by (ix1,ix2).
- virtual const double & [get](#) (size_t ix1, size_t ix2) const
Get the element indexed by (ix1,ix2).
- virtual int [set](#) (size_t ix1, size_t ix2, double val)
Set the element indexed by (ix1,ix2) to value val.
- virtual double & [operator\(\)](#) (size_t ix, size_t iy)
Get the element indexed by (ix1,ix2).

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

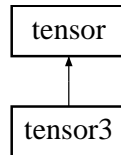
- [tensor.h](#)

8.369 tensor3 Class Reference

Rank 3 [tensor](#).

```
#include <tensor.h>
```

Inheritance diagram for tensor3::



8.369.1 Detailed Description

Rank 3 [tensor](#).

Definition at line 914 of file tensor.h.

Public Member Functions

- [tensor3](#) ()
Create an empty [tensor](#).
- [tensor3](#) (size_t sz, size_t sz2, size_t sz3)
Create a rank 3 [tensor](#) of size (sz,sz2,sz3).
- virtual double & [get](#) (size_t ix1, size_t ix2, size_t ix3)
Get the element indexed by (ix1,ix2,ix3).
- virtual const double & [get](#) (size_t ix1, size_t ix2, size_t ix3) const
Get the element indexed by (ix1,ix2,ix3).
- virtual int [set](#) (size_t ix1, size_t ix2, size_t ix3, double val)
Set the element indexed by (ix1,ix2,ix3) to value val.

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

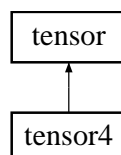
- [tensor.h](#)

8.370 tensor4 Class Reference

Rank 4 [tensor](#).

```
#include <tensor.h>
```

Inheritance diagram for tensor4::



8.370.1 Detailed Description

Rank 4 [tensor](#).

Definition at line 1010 of file tensor.h.

Public Member Functions

- `tensor4()`
Create an empty `tensor`.
- `tensor4(size_t sz, size_t sz2, size_t sz3, size_t sz4)`
Create a rank 4 `tensor` of size $(sz, sz2, sz3, sz4)$.
- virtual `double & get(size_t ix1, size_t ix2, size_t ix3, size_t ix4)`
Get the element indexed by $(ix1, ix2, ix3, ix4)$.
- virtual `const double & get(size_t ix1, size_t ix2, size_t ix3, size_t ix4) const`
Get the element indexed by $(ix1, ix2, ix3, ix4)$.
- virtual `int set(size_t ix1, size_t ix2, size_t ix3, size_t ix4, double val)`
Set the element indexed by $(ix1, ix2, ix3, ix4)$ to value `val`.

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

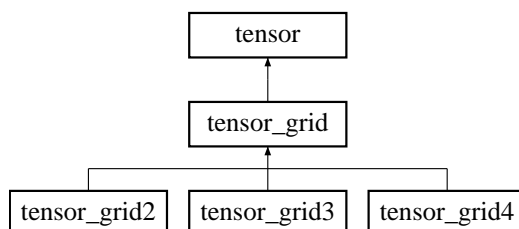
- [tensor.h](#)

8.371 `tensor_grid` Class Reference

Tensor class with arbitrary dimensions with a grid.

```
#include <tensor.h>
```

Inheritance diagram for `tensor_grid`:



8.371.1 Detailed Description

Tensor class with arbitrary dimensions with a grid.

This `tensor` class allows one to assign the indexes to numerical scales, so that n-dimensional interpolation can be performed. To set the grid, use `set_grid()` and then interpolation can be done using `interpolate()`.

Idea for future

Only allocate space for grid if it is set

Idea for future

Could implement arithmetic operators + and - and some different products.

Definition at line 387 of file `tensor.h`.

Public Member Functions

- `tensor_grid()`
Create an empty `tensor` with zero rank.

- **tensor_grid** (size_t rank, size_t *dim)
*Create a **tensor** of rank rank with sizes given in dim.*
- virtual int **set_val** (double *grdp, double val)
Set the element closest to grid point grdp to value val.
- virtual int **set_val** (double *grdp, double val, double *closest)
Set the element closest to grid point grdp to value val.
- virtual double **get_val** (double *grdp)
Get the element closest to grid point grdp.
- virtual double **get_val** (double *grdp, double *closest)
Get the element closest to grid point grdp to value val.
- virtual int **set_grid** (double **val)
Set the grid.
- virtual int **tensor_allocate** (size_t rank, size_t *dim)
*Allocate space for a **tensor** of rank rank with sizes given in dim.*
- virtual int **tensor_free** ()
Free allocated space (also sets rank to zero).
- virtual size_t **lookup_grid** (size_t i, double val)
Lookup index for grid closest to val.
- virtual double **get_grid** (size_t i, size_t j)
Lookup index for grid closest to val.
- virtual int **lookup_grid** (double *vals, size_t *indices)
Lookup indices for grid closest to val.
- virtual size_t **lookup_grid_val** (size_t i, double val, double &val2)
Lookup index for grid closest to val, returning the grid point.
- int **set_interp** (base_interp_mgr< double * > &bi1, base_interp_mgr< array_const_subvector > &bi2)
Set interpolation managers.
- virtual double **interpolate** (double *vals)
*Interpolate values vals into the **tensor**, returning the result.*

Data Fields

Default interpolation managers

- **def_interp_mgr**< double *, **cspline_interp** > **dim1**
- **def_interp_mgr**< **array_const_subvector**, **cspline_interp** > **dim2**

Protected Attributes

- double ** **grd**
A rank-sized set of arrays for the grid points.
- bool **grid_set**
If true, the grid has been set by the user.
- **base_interp_mgr**< double * > * **bim1**
The interpolation manager.
- **base_interp_mgr**< **array_const_subvector** > * **bim2**
The subvector interpolation manager.

8.371.2 Constructor & Destructor Documentation

8.371.2.1 **tensor_grid** (size_t rank, size_t *dim) [inline]

Create a **tensor** of rank rank with sizes given in dim.

The parameter dim must be a pointer to an array of sizes with length rank. If the user requests any of the sizes to be zero, this constructor will call the error handler, create an empty **tensor**, and will allocate no memory.

Idea for future

Create a "tensor_grid1" for completeness.

Definition at line 427 of file `tensor.h`.

8.371.3 Member Function Documentation

8.371.3.1 `virtual double interpolate (double * vals) [inline, virtual]`

Interpolate values `vals` into the [tensor](#), returning the result.

This is a quick and dirty implementation of n-dimensional interpolation by recursive application of the 1-dimensional routine from [smart_interp_vec](#), using the base interpolation object specified in the template parameter `base_interp_t`. This will be slow for sufficiently large data sets.

Idea for future

It should be straightforward to improve the scaling of this algorithm significantly by creating a "window" of local points around the point of interest. This could be done easily by constructing an initial subtensor.

Definition at line 705 of file `tensor.h`.

8.371.3.2 `virtual int set_grid (double ** val) [inline, virtual]`

Set the grid.

The parameter `grid` must define the grid, so that `val[i][j]` is the `j`th grid point for the `i`th index. The size of array `grid[i]` should be given by `dim[i]` where `dim` was the argument given in the constructor or to the function [tensor_allocate\(\)](#).

Note that the grid is copied so the function argument may be destroyed by the user after calling [set_grid\(\)](#).

Idea for future

Define a more generic interface for matrix types

Definition at line 558 of file `tensor.h`.

8.371.3.3 `virtual int tensor_allocate (size_t rank, size_t * dim) [inline, virtual]`

Allocate space for a [tensor](#) of rank `rank` with sizes given in `dim`.

The parameter `dim` must be a pointer to an array of sizes with length `rank`.

If memory was previously allocated, it will be freed before the new allocation and previously specified grid data will be lost.

If the user requests any of the sizes to be zero, this function will call the error handler and will allocate no memory. If memory was previously allocated, the [tensor](#) is left unmodified and no deallocation is performed.

Reimplemented from [tensor](#).

Definition at line 584 of file `tensor.h`.

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

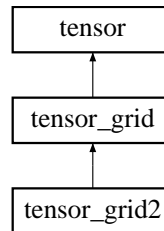
- [tensor.h](#)

8.372 `tensor_grid2` Class Reference

Rank 2 [tensor](#) with a grid.

```
#include <tensor.h>
```

Inheritance diagram for `tensor_grid2`:



8.372.1 Detailed Description

Rank 2 [tensor](#) with a grid.

Definition at line 859 of file `tensor.h`.

Public Member Functions

- [tensor_grid2](#) ()
Create an empty [tensor](#).
- [tensor_grid2](#) (size_t sz, size_t sz2)
Create a rank 2 [tensor](#) of size (sz,sz2,sz3).
- virtual double & [get](#) (size_t ix1, size_t ix2)
Get the element indexed by (ix1,ix2,ix3).
- virtual const double & [get](#) (size_t ix1, size_t ix2) const
Get the element indexed by (ix1,ix2,ix3).
- virtual int [set](#) (size_t ix1, size_t ix2, double val)
Set the element indexed by (ix1,ix2,ix3) to value val.

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

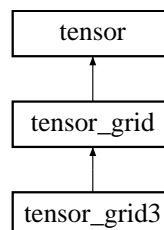
- [tensor.h](#)

8.373 **tensor_grid3** Class Reference

Rank 3 [tensor](#) with a grid.

```
#include <tensor.h>
```

Inheritance diagram for `tensor_grid3`:



8.373.1 Detailed Description

Rank 3 `tensor` with a grid.

Definition at line 953 of file `tensor.h`.

Public Member Functions

- `tensor_grid3` ()
Create an empty `tensor`.
- `tensor_grid3` (size_t sz, size_t sz2, size_t sz3)
Create a rank 3 `tensor` of size (sz,sz2,sz3).
- virtual double & `get` (size_t ix1, size_t ix2, size_t ix3)
Get the element indexed by (ix1,ix2,ix3).
- virtual const double & `get` (size_t ix1, size_t ix2, size_t ix3) const
Get the element indexed by (ix1,ix2,ix3).
- virtual int `set` (size_t ix1, size_t ix2, size_t ix3, double val)
Set the element indexed by (ix1,ix2,ix3) to value val.

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

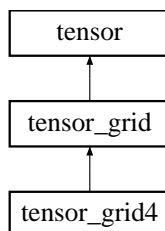
- `tensor.h`

8.374 `tensor_grid4` Class Reference

Rank 4 `tensor` with a grid.

```
#include <tensor.h>
```

Inheritance diagram for `tensor_grid4`:



8.374.1 Detailed Description

Rank 4 `tensor` with a grid.

Definition at line 1054 of file `tensor.h`.

Public Member Functions

- `tensor_grid4` ()
Create an empty `tensor`.
- `tensor_grid4` (size_t sz, size_t sz2, size_t sz3, size_t sz4)
Create a rank 4 `tensor` of size (sz,sz2,sz3,sz4).
- virtual double & `get` (size_t ix1, size_t ix2, size_t ix3, size_t ix4)
Get the element indexed by (ix1,ix2,ix3,ix4).
- virtual const double & `get` (size_t ix1, size_t ix2, size_t ix3, size_t ix4) const

Get the element indexed by (ix1,ix2,ix3,ix4).

- virtual int [set](#) (size_t ix1, size_t ix2, size_t ix3, size_t ix4, double val)
Set the element indexed by (ix1,ix2,ix3,ix4) to value val.

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

- [tensor.h](#)

8.375 test_mgr Class Reference

A class to manage testing and record success and failure.

```
#include <test_mgr.h>
```

8.375.1 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

- 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, class vec2_t >
bool [test_rel_arr](#) (int nv, vec_t &result, vec2_t &expected, double rel_error, std::string description)
Test for $|result - expected|/expected < rel_error$ over each element of an array.
- template<class mat_t, class mat2_t >
bool [test_rel_mat](#) (int nr, int nc, mat_t &result, mat2_t &expected, double rel_error, std::string description)
Test for $|result - expected|/expected < rel_error$ over each element of an array.
- template<class vec_t, class vec2_t >
bool [test_abs_arr](#) (int nv, vec_t &result, vec2_t &expected, double rel_error, std::string description)

Test for $|\text{result} - \text{expected}| / < \text{abs_error}$ over each element of an array.

- template<class vec_t, class vec2_t >
bool [test_fact_arr](#) (int nv, vec_t &result, vec2_t &expected, double factor, std::string description)
Test for $1/\text{factor} < \text{result}/\text{expected} < \text{factor}$ over each element of an array.
- template<class vec_t >
bool [test_gen_arr](#) (int nv, vec_t &result, vec_t &expected, std::string description)
Test for equality of a generic array.

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).

8.375.2 Member Function Documentation

8.375.2.1 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.

8.375.2.2 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:

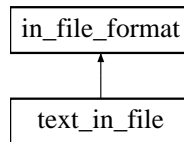
- test_mgr.h

8.376 text_in_file Class Reference

An input text file.

```
#include <text_file.h>
```

Inheritance diagram for text_in_file::



8.376.1 Detailed Description

An input text file.

This class is experimental.

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 220 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:

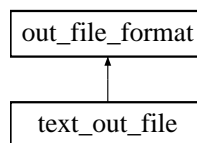
- text_file.h

8.377 text_out_file Class Reference

An output text file.

```
#include <text_file.h>
```

Inheritance diagram for text_out_file::



8.377.1 Detailed Description

An output text file.

This class is experimental.

Todo

Test output with <'s and >'s and document this

Todo

Document the difference between `flush()` and `end_line()`

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 49 of file text_file.h.

Public Member Functions

- [text_out_file](#) (std::ostream *out_file, int width=80, bool bracket_objs=true)
Use output stream out_file for text output.
- [text_out_file](#) (std::string file_name, std::ostream *prop=0, bool append=false, int width=80, bool bracket_objs=true)
Create an output file with name file_name.
- virtual int [open](#) (std::ostream *out_file, int width=80, bool bracket_objs=true)
Desc.
- virtual int [open](#) (std::string file_name, std::ostream *prop=0, bool append=false, int width=80, bool bracket_objs=true)
Desc.
- virtual int [close](#) ()
Desc.
- 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](#) ()
Desc.
- 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 if the current string is long enough.
- bool [is_hc_type](#) (std::string type)
If true, then type is a "hard-coded" type.

Protected Attributes

- bool [extra_brackets](#)
Desc.
- 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.

8.377.2 Constructor & Destructor Documentation

8.377.2.1 text_out_file (std::ostream * out_file, int width = 80, bool bracket_objs = true)

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.

8.377.2.2 text_out_file (std::string file_name, std::ostream * prop = 0, bool append = false, int width = 80, bool bracket_objs = true)

Create an output file with name `file_name`.

If `prop` is not NULL (i.e. nonzero), then the I/O properties (precision, fill, flags, etc) for the newly created file are taken to be the same as `prop`.

8.377.3 Member Function Documentation

8.377.3.1 virtual int end_line () [virtual]

End line.

This calls `flush()` in case the current string buffer is larger than the width, and then outputs the remaining characters and creates a new stringstream.

Implements `out_file_format`.

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

- `text_file.h`

8.378 twod_eqi_intp Class Reference

Two-dimensional interpolation for equally-spaced intervals.

```
#include <twod_eqi_intp.h>
```

8.378.1 Detailed Description

Two-dimensional interpolation for equally-spaced intervals.

Note:

This class is unfinished.

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 57 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.

8.378.2 Member Function Documentation

8.378.2.1 int set_type (int type) [inline]

Set the interpolation type.

- 3: 3-point
 - 4: 4-point
 - 6: 6-point (default)
-

Definition at line 80 of file twod_eqi_intp.h.

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

- twod_eqi_intp.h

8.379 twod_intp Class Reference

Two-dimensional interpolation class.

```
#include <twod_intp.h>
```

8.379.1 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 of the matrix to be specified in the function `set_data()` and this function is designed to follow the format:

	x_0	x_1	x_2
y_0	M_{00}	M_{01}	M_{02}
y_1	M_{10}	M_{11}	M_{12}
y_2	M_{20}	M_{21}	M_{22}

thus the matrix should be $M[i][j]$ where i is the y index and j is the row index. (See also the discussion in the User's guide in the section called [Rows and columns vs. x and y.](#))

The function `set_data()`, does not copy the data, it stores pointers to the data. If the data is modified, then the function `reset_interp()` must be called to reset the interpolation information with the original pointer information. The storage for the data, including the arrays `x_grid` and `y_grid` are all managed by the user.

By default, cubic spline interpolation with natural boundary conditions is used. This can be changed with the `set_interp()` function.

There is an example for the usage of this class given in `examples/ex_twod_intp.cpp`.

Idea for future

Could also include mixed second/first derivatives: `deriv_xxy` and `deriv_xyy`.

Idea for future

Implement an improved caching system in case, for example `xfirst` is true and the last interpolation used the same value of `x`.

Definition at line 100 of file twod_intp.h.

Public Member Functions

- int `set_data` (size_t n_x, size_t n_y, [ovector](#) &x_grid, [ovector](#) &y_grid, [omatrix](#) &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 $\frac{\partial^2 f}{\partial x \partial y}$.
- int [set_interp](#) (base_interp_mgr< ovector_const_view > &b1, base_interp_mgr< ovector_const_subvector > &b2)
Specify the base interpolation objects to use.
- int [unset_data](#) ()
Inform the class the data has been modified or changed in a way that [set_data\(\)](#) will need to be called again.

8.379.2 Member Function Documentation

8.379.2.1 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

8.379.2.2 int set_data (size_t n_x, size_t n_y, ovector & x_grid, ovector & y_grid, omatrix & 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 may give slightly different results.

8.379.2.3 int set_interp (base_interp_mgr< ovector_const_view > &b1, base_interp_mgr< ovector_const_subvector > &b2) [inline]

Specify the base interpolation objects to use.

This allows the user to provide new interpolation objects for use in the two-dimensional interpolation. For example,

```
twod_intp ti;
ovector x(20), y(40);
omatrix d(40,20);

// Fill x, y, and d with the data, choose linear interpolation
// instead of the default cubic spline
def_interp_mgr<ovector_const_view,linear_interp> dim1;
def_interp_mgr<ovector_const_subvector,linear_interp> dim2;

ti.set_interp(dim1,dim2);
ti.set_data(20,40,x,y,d,true);
```

This function automatically calls [reset_interp\(\)](#) and [reset_data\(\)](#) if the data has already been set to reset the internal interpolation objects.

Definition at line 201 of file twod_intp.h.

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

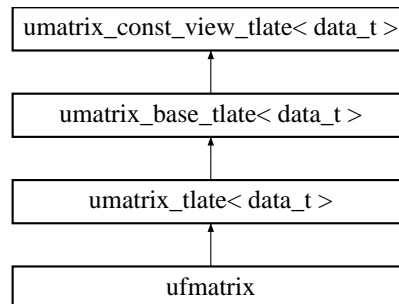
- [twod_intp.h](#)

8.380 **ufmatrix Class Template Reference**

A matrix where the memory allocation is performed in the constructor.

```
#include <ufmatrix_tlate.h>
```

Inheritance diagram for `ufmatrix::`



8.380.1 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 932 of file `ufmatrix_tlate.h`.

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

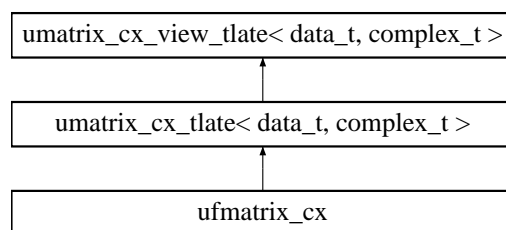
- [ufmatrix_tlate.h](#)

8.381 **ufmatrix_cx Class Template Reference**

A matrix where the memory allocation is performed in the constructor.

```
#include <ufmatrix_cx_tlate.h>
```

Inheritance diagram for `ufmatrix_cx::`



8.381.1 Detailed Description

template<size_t N, size_t M> class ufmatrix_cx< N, M >

A matrix where the memory allocation is performed in the constructor.

Definition at line 716 of file umatrix_cx_tlate.h.

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

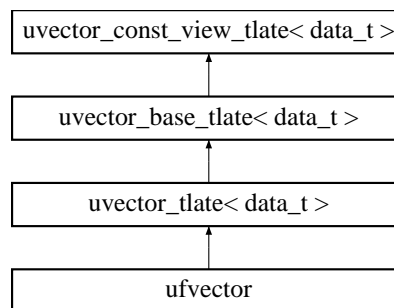
- [umatrix_cx_tlate.h](#)

8.382 uvector Class Template Reference

A vector with unit-stride where the memory allocation is performed in the constructor.

`#include <uvector_tlate.h>`

Inheritance diagram for uvector::



8.382.1 Detailed Description

template<size_t N = 0> class uvector< N >

A vector with unit-stride where the memory allocation is performed in the constructor.

Definition at line 1033 of file uvector_tlate.h.

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

- [uvector_tlate.h](#)

8.383 umatrix_alloc Class Reference

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

`#include <umatrix_tlate.h>`

8.383.1 Detailed Description

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

Definition at line 920 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:

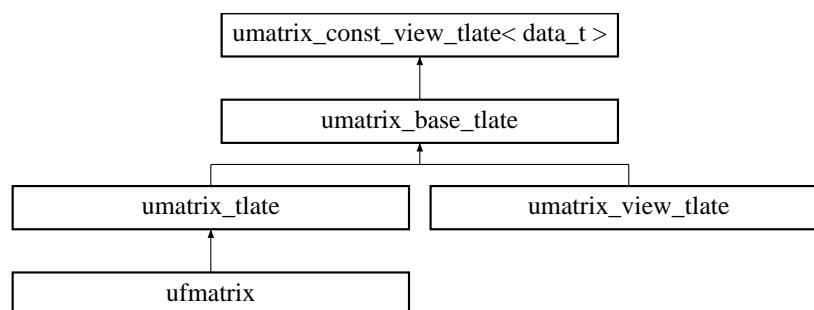
- [umatrix_tlate.h](#)

8.384 umatrix_base_tlate Class Template Reference

A matrix view of double-precision numbers.

```
#include <umatrix_tlate.h>
```

Inheritance diagram for umatrix_base_tlate::



8.384.1 Detailed Description

```
template<class data_t> class umatrix_base_tlate< data_t >
```

A matrix view of double-precision numbers.

Definition at line 206 of file umatrix_tlate.h.

Public Member Functions

Copy constructors

- [umatrix_base_tlate](#) (umatrix_base_tlate &v)
Shallow copy constructor - create a new view of the same matrix.
- [umatrix_base_tlate](#) & [operator=](#) (umatrix_base_tlate &v)
Shallow 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_ptr` (`size_t i`, `size_t j`)
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`.

Arithmetic

- `umatrix_base_tlate< data_t > & operator+=` (`const umatrix_base_tlate< data_t > &x`)
operator+=
- `umatrix_base_tlate< data_t > & operator-=` (`const umatrix_base_tlate< data_t > &x`)
operator-=
- `umatrix_base_tlate< data_t > & operator+=` (`const data_t &y`)
operator+=
- `umatrix_base_tlate< data_t > & operator-=` (`const data_t &y`)
operator-=
- `umatrix_base_tlate< data_t > & operator*=` (`const data_t &y`)
operator=*

Protected Member Functions

- `umatrix_base_tlate` ()
Empty constructor.

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

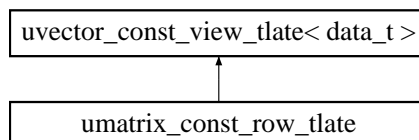
- [umatrix_tlate.h](#)

8.385 `umatrix_const_row_tlate` Class Template Reference

Create a const vector from a row of a matrix.

```
#include <umatrix_tlate.h>
```

Inheritance diagram for `umatrix_const_row_tlate`:



8.385.1 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 834 of file `umatrix_tlate.h`.

Public Member Functions

- `umatrix_const_row_tlate` (const `umatrix_base_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:

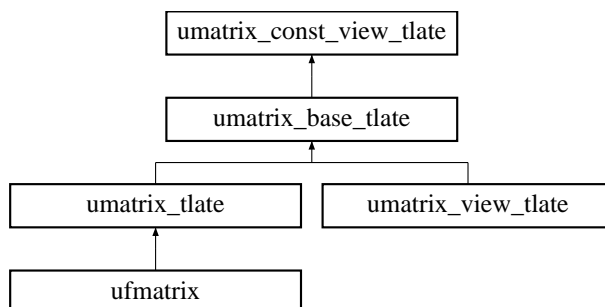
- `umatrix_tlate.h`

8.386 `umatrix_const_view_tlate` Class Template Reference

A matrix view of double-precision numbers.

```
#include <umatrix_tlate.h>
```

Inheritance diagram for `umatrix_const_view_tlate`:



8.386.1 Detailed Description

```
template<class data_t> class umatrix_const_view_tlate< data_t >
```

A matrix view of double-precision numbers.

Definition at line 49 of file `umatrix_tlate.h`.

Public Member Functions

Copy constructors

- `umatrix_const_view_tlate` (const `umatrix_const_view_tlate` &v)
Shallow copy constructor - create a new view of the same matrix.
- `umatrix_const_view_tlate` & `operator=` (const `umatrix_const_view_tlate` &v)
Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- const data_t * `operator[]` (size_t i) const
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).
- const data_t * `get_const_ptr` (size_t i, size_t j) const
Get pointer (with optional range-checking).
- 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.

Protected Member Functions

- `umatrix_const_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.

8.386.2 Member Function Documentation

8.386.2.1 `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 176 of file `umatrix_tlate.h`.

8.386.2.2 `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 166 of file `umatrix_tlate.h`.

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

- `umatrix_tlate.h`

8.387 **umatrix_cx_alloc Class Reference**

A simple class to provide an `allocate ()` function for `umatrix_cx`.

```
#include <umatrix_cx_tlate.h>
```

8.387.1 Detailed Description

A simple class to provide an `allocate ()` function for `umatrix_cx`.

Definition at line 704 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:

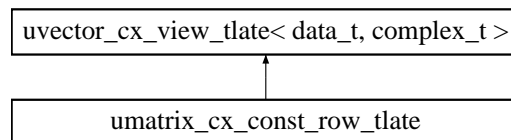
- [umatrix_cx_tlate.h](#)

8.388 `umatrix_cx_const_row_tlate` Class Template Reference

Create a const vector from a row of a matrix.

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for `umatrix_cx_const_row_tlate`:



8.388.1 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 632 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:

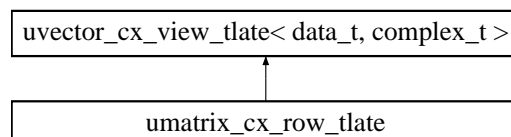
- [umatrix_cx_tlate.h](#)

8.389 `umatrix_cx_row_tlate` Class Template Reference

Create a vector from a row of a matrix.

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for `umatrix_cx_row_tlate`:



8.389.1 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 616 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:

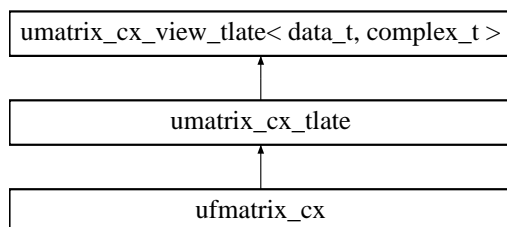
- [`umatrix_cx_tlate.h`](#)

8.390 `umatrix_cx_tlate` Class Template Reference

A matrix of double-precision numbers.

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for `umatrix_cx_tlate`:



8.390.1 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 374 of file `umatrix_cx_tlate.h`.

Public Member Functions

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 nrow, size_t ncol)
Allocate memory after freeing any memory presently in use.
- int `free` ()
Free the memory.

Other methods

- `umatrix_cx_tlate`< data_t, complex_t > `transpose` ()
Compute the transpose (even if matrix is not square).

8.390.2 Member Function Documentation

8.390.2.1 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 587 of file `umatrix_cx_tlate.h`.

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

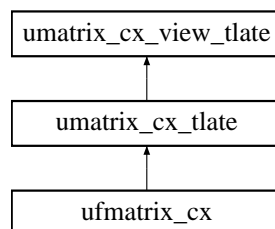
- `umatrix_cx_tlate.h`

8.391 `umatrix_cx_view_tlate` Class Template Reference

A matrix view of complex numbers.

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for `umatrix_cx_view_tlate`:



8.391.1 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

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.

8.391.2 Member Function Documentation

8.391.2.1 `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 275 of file `umatrix_cx_tlate.h`.

8.391.2.2 `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 265 of file `umatrix_cx_tlate.h`.

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

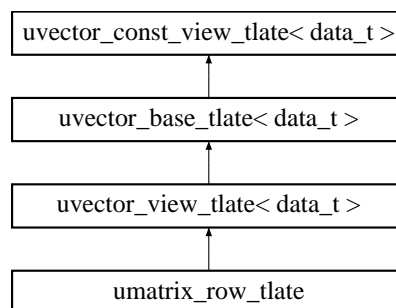
- `umatrix_cx_tlate.h`

8.392 `umatrix_row_tlate` Class Template Reference

Create a vector from a row of a matrix.

```
#include <umatrix_tlate.h>
```

Inheritance diagram for `umatrix_row_tlate`:



8.392.1 Detailed Description

```
template<class data_t> class umatrix_row_tlate< data_t >
```

Create a vector from a row of a matrix.

Definition at line 816 of file `umatrix_tlate.h`.

Public Member Functions

- [`umatrix_row_tlate`](#) ([`umatrix_base_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:

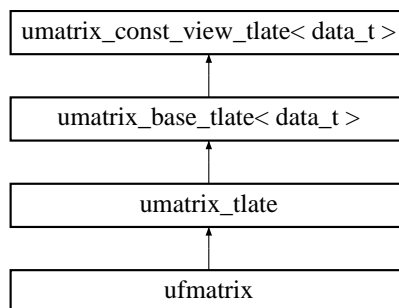
- [`umatrix_tlate.h`](#)

8.393 `umatrix_tlate` Class Template Reference

A matrix of double-precision numbers.

```
#include <umatrix_tlate.h>
```

Inheritance diagram for `umatrix_tlate`::



8.393.1 Detailed Description

```
template<class data_t> class umatrix_tlate< data_t >
```

A matrix of double-precision numbers.

Definition at line 575 of file `umatrix_tlate.h`.

Public Member Functions

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 nrow`, `size_t ncol`)
Allocate memory after freeing any memory presently in use.
- `int free` ()
Free the memory.

Other methods

- `umatrix_tlate< data_t > transpose` ()
Compute the transpose (even if matrix is not square).

8.393.2 Member Function Documentation

8.393.2.1 `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 787 of file `umatrix_tlate.h`.

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

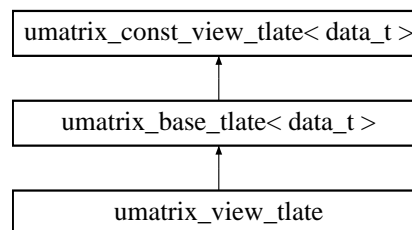
- [umatrix_tlate.h](#)

8.394 `umatrix_view_tlate` Class Template Reference

A matrix view of double-precision numbers.

```
#include <umatrix_tlate.h>
```

Inheritance diagram for `umatrix_view_tlate`:



8.394.1 Detailed Description

```
template<class data_t> class umatrix_view_tlate< data_t >
```

A matrix view of double-precision numbers.

Definition at line 433 of file `umatrix_tlate.h`.

Public Member Functions

Copy constructors

- `umatrix_view_tlate` (const `umatrix_view_tlate` &v)
Shallow copy constructor - create a new view of the same matrix.
- `umatrix_view_tlate` & `operator=` (const `umatrix_view_tlate` &v)
Shallow copy constructor - create a new view of the same matrix.
- `umatrix_view_tlate` (`umatrix_base_tlate`< data_t > &v)
Shallow copy constructor - create a new view of the same matrix.
- `umatrix_view_tlate` & `operator=` (`umatrix_base_tlate`< data_t > &v)
Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- `data_t` * `operator[]` (size_t i) const
Array-like indexing.
- `data_t` & `operator()` (size_t i, size_t j) const
Array-like indexing.
- `data_t` * `get_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) const
Set (with optional range-checking).
- int `set_all` (double val) const
Set all of the value to be the value val.

Protected Member Functions

- `umatrix_view_tlate` ()
Empty constructor.

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

- `umatrix_tlate.h`

8.395 `uvector_alloc` Class Reference

A simple class to provide an `allocate()` function for `uvector`.

```
#include <uvector_tlate.h>
```

8.395.1 Detailed Description

A simple class to provide an `allocate()` function for `uvector`.

Definition at line 1010 of file `uvector_tlate.h`.

Public Member Functions

- void `allocate` (`uvector` &o, size_t i)
Allocate v for i elements.
- void `free` (`uvector` &o)
Free memory.

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

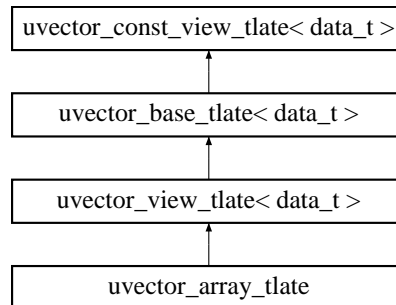
- `uvector_tlate.h`

8.396 `uvector_array_tlate` Class Template Reference

Create a vector from an array.

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_array_tlate`::



8.396.1 Detailed Description

```
template<class data_t> class uvector_array_tlate< data_t >
```

Create a vector from an array.

Definition at line 863 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:

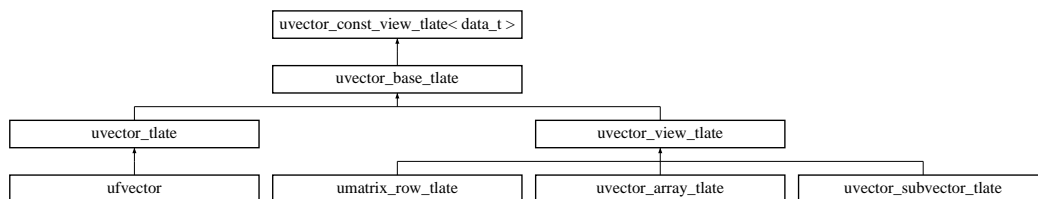
- [`uvector_tlate.h`](#)

8.397 `uvector_base_tlate` Class Template Reference

Desc.

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_base_tlate`::



8.397.1 Detailed Description

`template<class data_t> class uvector_base_tlate< data_t >`

Desc.

Definition at line 274 of file `uvector_tlate.h`.

Public Member Functions

Copy constructors

- `uvector_base_tlate` (`uvector_base_tlate` &`v`)
Copy constructor - create a new view of the same vector.
- `uvector_base_tlate` & `operator=` (`uvector_base_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_ptr` (`size_t` `i`)
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`.

Arithmetic

- `uvector_base_tlate< data_t >` & `operator+=` (`const uvector_base_tlate< data_t >` &`x`)
operator+=
- `uvector_base_tlate< data_t >` & `operator-=` (`const uvector_base_tlate< data_t >` &`x`)
operator-=
- `uvector_base_tlate< data_t >` & `operator+=` (`const data_t` &`y`)
operator+=
- `uvector_base_tlate< data_t >` & `operator-=` (`const data_t` &`y`)
operator-=
- `uvector_base_tlate< data_t >` & `operator*=` (`const data_t` &`y`)
operator=*

Protected Member Functions

- `uvector_base_tlate` ()
Empty constructor for use by children.

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

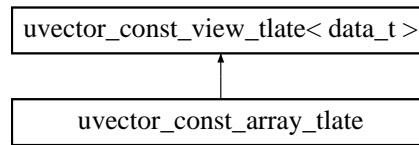
- `uvector_tlate.h`

8.398 `uvector_const_array_tlate` Class Template Reference

Create a vector from an const array.

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_const_array_tlate`::



8.398.1 Detailed Description

```
template<class data_t> class uvector_const_array_tlate< data_t >
```

Create a vector from an const array.

Definition at line 897 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`.

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

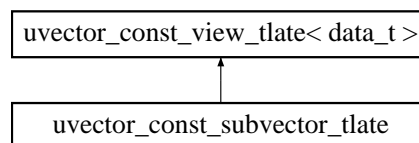
- [`uvector_tlate.h`](#)

8.399 `uvector_const_subvector_tlate` Class Template Reference

Create a const vector from a subvector of another vector.

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_const_subvector_tlate`::



8.399.1 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 917 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:

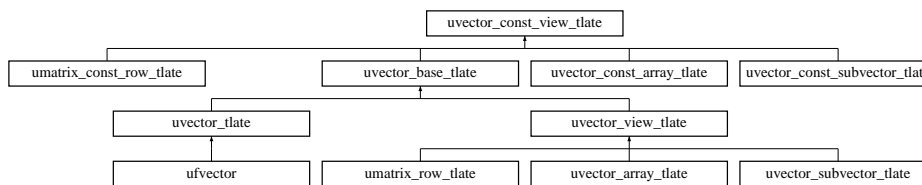
- `uvector_tlate.h`

8.400 `uvector_const_view_tlate` Class Template Reference

A const vector view with unit stride.

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_const_view_tlate`::



8.400.1 Detailed Description

```
template<class data_t> class uvector_const_view_tlate< data_t >
```

A const vector view with unit stride.

Idea for future

Could allow user-defined specification of restrict keyword

Definition at line 61 of file `uvector_tlate.h`.

Public Member Functions

- `data_t norm` () const
Norm.

Copy constructors

- `uvector_const_view_tlate` (const `uvector_const_view_tlate` &v)
Copy constructor - create a new view of the same vector.
- `uvector_const_view_tlate &operator=` (const `uvector_const_view_tlate` &v)
Copy constructor - create a new view of the same vector.

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).

- `size_t size () const`

Method to return vector size.

Other methods

- `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.

Protected Member Functions

- `uvector_const_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.

8.400.2 Member Function Documentation

8.400.2.1 `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 `O2SCL_ERR()` and returns 0.

If more than one entry is the same distance from `x0`, this function returns the entry with smallest index.

Definition at line 198 of file `uvector_tlate.h`.

8.400.2.2 `size_t size () const` `[inline]`

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 174 of file `uvector_tlate.h`.

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

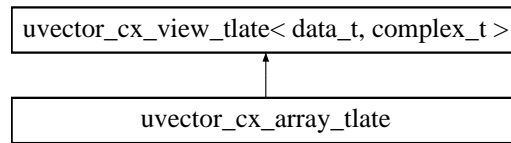
- `uvector_tlate.h`

8.401 `uvector_cx_array_tlate` Class Template Reference

Create a vector from an array.

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_array_tlate`::



8.401.1 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 497 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:

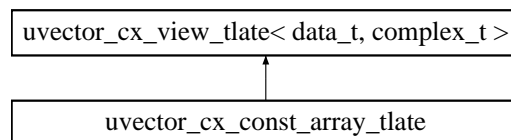
- [`uvector_cx_tlate.h`](#)

8.402 `uvector_cx_const_array_tlate` Class Template Reference

Create a vector from an array.

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_const_array_tlate`::



8.402.1 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 531 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.

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 `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:

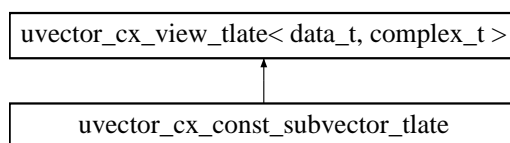
- [uvector_cx_tlate.h](#)

8.403 `uvector_cx_const_subvector_tlate` Class Template Reference

Create a vector from a subvector of another.

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_const_subvector_tlate`:



8.403.1 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 579 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:

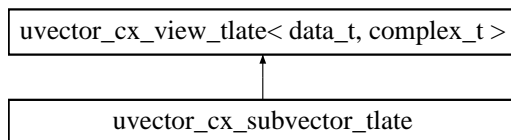
- [uvector_cx_tlate.h](#)

8.404 `uvector_cx_subvector_tlate` Class Template Reference

Create a vector from a subvector of another.

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_subvector_tlate`:



8.404.1 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 512 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:

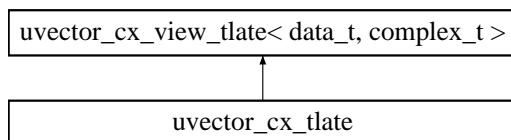
- [`uvector_cx_tlate.h`](#)

8.405 `uvector_cx_tlate` Class Template Reference

A vector of double-precision numbers with unit stride.

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_tlate`:



8.405.1 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 is also an << operator for this class documented "Functions" section of [`uvector_tlate.h`](#).

Definition at line 320 of file `uvector_cx_tlate.h`.

Public Member Functions

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.

8.405.2 Member Function Documentation

8.405.2.1 `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 482 of file `uvector_cx_tlate.h`.

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

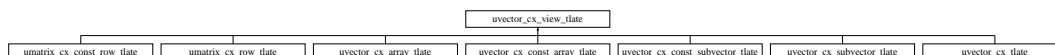
- [uvector_cx_tlate.h](#)

8.406 `uvector_cx_view_tlate` Class Template Reference

A vector view of complex numbers with unit stride.

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for `uvector_cx_view_tlate`:



8.406.1 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.

Idea for future

Write `lookup()` method, and possibly an `erase()` method.

Definition at line 48 of file `uvector_cx_tlate.h`.

Public Member Functions

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

- 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.

8.406.2 Member Function Documentation

8.406.2.1 `size_t size () const` [inline]

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 234 of file `uvector_cx_tlate.h`.

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

- `uvector_cx_tlate.h`

8.407 `uvector_int_alloc` Class Reference

A simple class to provide an `allocate ()` function for `uvector_int`.

```
#include <uvector_tlate.h>
```

8.407.1 Detailed Description

A simple class to provide an `allocate ()` function for `uvector_int`.

Definition at line 1021 of file `uvector_tlate.h`.

Public Member Functions

- `void allocate (uvector_int &o, size_t i)`
Allocate \forall for i elements.
- `void free (uvector_int &o)`
Free memory.

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

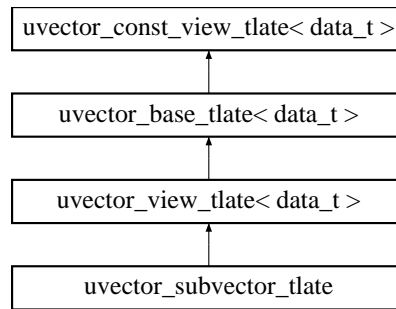
- `uvector_tlate.h`

8.408 `uvector_subvector_tlate` Class Template Reference

Create a vector from a subvector of another.

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_subvector_tlate::`



8.408.1 Detailed Description

template<class data_t> class `uvector_subvector_tlate< data_t >`

Create a vector from a subvector of another.

Definition at line 878 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:

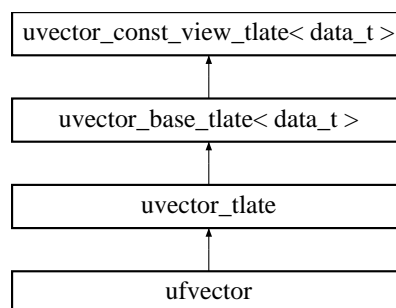
- [`uvector_tlate.h`](#)

8.409 `uvector_tlate` Class Template Reference

A vector with unit stride.

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_tlate`:



8.409.1 Detailed Description

template<class data_t> class `uvector_tlate< data_t >`

A vector with unit stride.

There are several useful methods which are defined in the parent class, `uvector_view_tlate`. There is also an `<<` operator for this class documented "Functions" section of `uvector_tlate.h`.

Definition at line 616 of file `uvector_tlate.h`.

Public Member Functions

- `int sort_unique ()`
Sort the vector and ensure all elements are unique by removing duplicates.

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_const_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_const_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.

8.409.2 Member Function Documentation

8.409.2.1 `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 795 of file `uvector_tlate.h`.

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

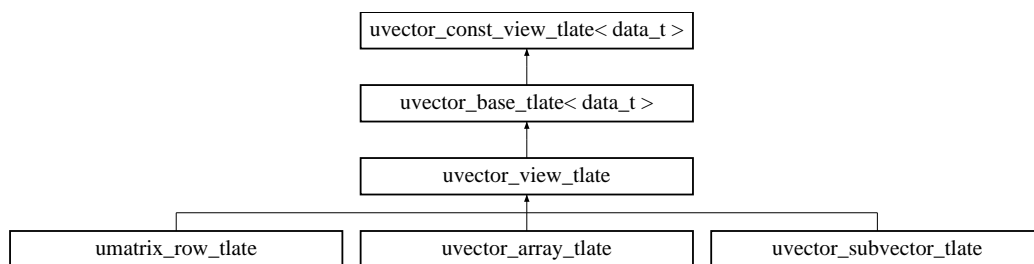
- `uvector_tlate.h`

8.410 `uvector_view_tlate` Class Template Reference

A base class for uvectors.

```
#include <uvector_tlate.h>
```

Inheritance diagram for `uvector_view_tlate::`



8.410.1 Detailed Description

`template<class data_t> class uvector_view_tlate< data_t >`

A base class for uvectors.

Definition at line 477 of file `uvector_tlate.h`.

Public Member Functions

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.
- `uvector_view_tlate` (`uvector_base_tlate`< *data_t* > &*v*)
Copy constructor - create a new view of the same vector.
- `uvector_view_tlate` & `operator=` (`uvector_base_tlate`< *data_t* > &*v*)
Copy constructor - create a new view of the same vector.

Get and set methods

- *data_t* & `operator[]` (*size_t i*) const
Array-like indexing.
- *data_t* & `operator()` (*size_t i*) const
Array-like indexing.
- *data_t* * `get_ptr` (*size_t i*) const
Get pointer (with optional range-checking).
- int `set` (*size_t i*, *data_t val*) const
Set (with optional range-checking).
- int `set_all` (*data_t val*) const
Set all of the value to be the value val.

Protected Member Functions

- `uvector_view_tlate` ()
Empty constructor provided for use by `uvector_view_tlate(const uvector_view_tlate &v)`.

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

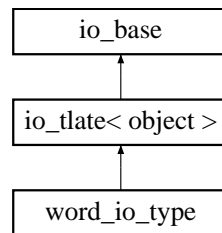
- `uvector_tlate.h`

8.411 word_io_type Class Reference

I/O object for words.

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

Inheritance diagram for word_io_type::



8.411.1 Detailed Description

I/O object for words.

This class is experimental.

Definition at line 2383 of file collection.h.

Public Member Functions

- [word_io_type](#) (const char *t)
Desc.
- int [input](#) (cinput *co, in_file_format *ins, std::string *dp)
Desc.
- int [output](#) (coutput *co, out_file_format *outs, std::string *dp)
Desc.
- const char * [type](#) ()
Desc.

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

- [collection.h](#)

9 File Documentation

9.1 array.h File Reference

Various array classes.

```
#include <iostream>
```

```
#include <cmath>
```

```
#include <string>
```

```
#include <fstream>
```

```
#include <sstream>
```

```
#include <new>
```

```
#include <o2scl/err_hnd.h>
#include <gsl/gsl_ieee_utils.h>
#include <gsl/gsl_sort.h>
```

9.1.1 Detailed Description

Various array classes.

For a more general discussion of vectors and matrices in `O2scl`, see the [Arrays, Vectors, Matrices and Tensors](#) of the User's Guide.

This file contains classes and functions for operating with C-style 1- or 2-dimensional arrays and pointers to double. For an example of the usage of the array allocation classes, see the [Multidimensional solver example](#). For more generic operations on generic vector objects (including in some cases C-style arrays), see also the file [vector.h](#).

This file contains the allocation classes

- [array_alloc](#)
- [array_2d_alloc](#)
- [pointer_alloc](#)
- [pointer_2d_alloc](#)

the classes for the manipulation of arrays 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` and `omatrix_col` (see usage in `src/ode/ode_it_solve_ts.cpp`)

- [array_2d_row](#)
- [array_2d_col](#)

Note:

The classes

- [array_reverse](#)
- [array_subvector](#)
- [array_subvector_reverse](#)
- [array_const_reverse](#)
- [array_const_subvector](#)
- [array_const_subvector_reverse](#)

can be used with pointers or arrays, but [array_alloc](#) and [pointer_alloc](#) are *not* interchangeable.

Definition in file [array.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 [pointer_2d_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_2d_col](#)
Column of a 2d array.
- class [array_2d_row](#)
Row of a 2d array.
- 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.

Functions

- `template<class type >`
`type * new_array (size_t n)`
Create a new C-style 1-dimensional array.
- `template<class type >`
`void delete_array (type *t)`
Free the memory for a C-style 1-dimensional array.
- `template<class type >`
`type ** new_2d_array (size_t nr, size_t nc)`
Create a new C-style 2-dimensional array.
- `template<class type >`
`void delete_2d_array (type **t, size_t nr)`
Free the memory for a C-style 2-dimensional array.

9.1.2 Function Documentation

9.1.2.1 `void delete_2d_array (type ** t, size_t nr)` [inline]

Free the memory for a C-style 2-dimensional array.

This function never calls the error handler, but may create a segmentation fault if improperly called multiple times.

Definition at line 549 of file array.h.

9.1.2.2 `void delete_array (type * t)` [inline]

Free the memory for a C-style 1-dimensional array.

This function never calls the error handler, but may create a segmentation fault if improperly called multiple times.

Definition at line 510 of file array.h.

9.1.2.3 type new_2d_array (size_t nr, size_t nc) [inline]**

Create a new C-style 2-dimensional array.

This function is convenient because it automatically calls the O₂scl error handler in case the allocation fails, and will free the portions of the memory which were successfully allocated.

Definition at line 522 of file array.h.

9.1.2.4 type* new_array (size_t n) [inline]

Create a new C-style 1-dimensional array.

This function is convenient because it automatically calls the O₂scl error handler in case the allocation fails.

Definition at line 493 of file array.h.

9.2 cblas_base.h File Reference

O₂scl basic linear algebra function templates.

9.2.1 Detailed Description

O₂scl basic linear algebra function templates.

Idea for future

Convert to size_t and add float and complex versions

Definition in file [cblas_base.h](#).

Namespaces

- namespace [o2scl_cblas](#)
Namespace for O₂scl CBLAS function templates with operator[].

Enumerations

- enum [O2CBLAS_ORDER](#) { **O2cblasRowMajor** = 101, **O2cblasColMajor** = 102 }
Matrix order, either column-major or row-major.
- enum [O2CBLAS_TRANSPOSE](#) { **O2cblasNoTrans** = 111, **O2cblasTrans** = 112, **O2cblasConjTrans** = 113 }
Transpose operations.
- enum [O2CBLAS_UPLO](#) { **O2cblasUpper** = 121, **O2cblasLower** = 122 }
Upper- or lower-triangular.
- enum [O2CBLAS_DIAG](#) { **O2cblasNonUnit** = 131, **O2cblasUnit** = 132 }
Unit or generic diagonal.
- enum [O2CBLAS_SIDE](#) { **O2cblasLeft** = 141, **O2cblasRight** = 142 }
Left or right sided operation.

Functions

- template<class mat_t >
int [dgemm](#) (const enum O2CBLAS_ORDER Order, const enum O2CBLAS_TRANSPOSE TransA, const enum O2CBLAS_TRANSPOSE TransB, const int M, const int N, const int K, const double alpha, const mat_t &A, const mat_t &B, const double beta, mat_t &C)

Compute $y = \alpha \text{op}(A)x + \beta y$.

Standard BLAS functions

- `template<class vec_t, class vec2_t >`
void `daxpy` (const int N, const double alpha, const vec_t &X, vec2_t &Y)
Compute $y = \alpha x + y$.
- `template<class vec_t, class vec2_t >`
double `ddot` (const int N, const vec_t &X, const vec2_t &Y)
Compute $r = x \cdot y$.
- `template<class vec_t >`
void `dscal` (const int N, const double alpha, vec_t &X)
Compute $x = \alpha x$.
- `template<class vec_t >`
double `dnrm2` (const int N, const vec_t &X)
Compute the squared norm of the vector X.
- `template<class mat_t, class vec_t >`
int `dgemv` (const enum O2CBLAS_ORDER order, const enum O2CBLAS_TRANSPOSE TransA, const int M, const int N, const double alpha, const mat_t &A, const vec_t &X, const double beta, vec_t &Y)
Compute $y = \alpha \text{op}(A)x + \beta y$.
- `template<class mat_t, class vec_t >`
int `dtrsv` (const enum O2CBLAS_ORDER order, const enum O2CBLAS_UPLO Uplo, const enum O2CBLAS_TRANSPOSE TransA, const enum O2CBLAS_DIAG Diag, const int M, const int N, const mat_t &A, vec_t &X)
Compute $x = \text{op}(A)^{-1}x$.

Helper BLAS functions

- `template<class vec_t, class vec2_t >`
void `daxpy_subvec` (const int N, const double alpha, const vec_t &X, vec2_t &Y, const int ie)
Compute $x = \alpha x$ beginning with index ie and ending with index N-1.
- `template<class vec_t, class vec2_t >`
double `ddot_subvec` (const int N, const vec_t &X, const vec2_t &Y, const int ie)
Compute $r = x \cdot y$ beginning with index ie and ending with index N-1.
- `template<class vec_t >`
void `dscal_subvec` (const int N, const double alpha, vec_t &X, const int ie)
Compute $x = \alpha x$ beginning with index ie and ending with index N-1.
- `template<class vec_t >`
double `dnrm2_subvec` (const int N, const vec_t &X, const int ie)
Compute the squared norm of the vector X beginning with index ie and ending with index N-1.
- `template<class mat_t >`
double `dnrm2_subcol` (const mat_t &M, const size_t ir, const size_t ic, const size_t N)
Compute the squared norm of the last N rows of a column of a matrix.
- `template<class mat_t, class mat_subcol_t >`
double `dnrm2_subcol2` (const mat_t &M, const size_t ir, const size_t ic, const size_t N)
Desc.
- `template<class mat_t >`
void `dscal_subcol` (mat_t &A, const size_t ir, const size_t ic, const size_t n, const double alpha)
Compute $x = \alpha x$.
- `template<class mat_t, class vec_t >`
void `daxpy_hv_sub` (const int N, const double alpha, const mat_t &X, vec_t &Y, const int ie)
Compute $x = \alpha x$ for `householder_hv_sub()`.
- `template<class mat_t, class vec_t >`
double `ddot_hv_sub` (const int N, const mat_t &X, const vec_t &Y, const int ie)
Compute $r = x \cdot y$ for `householder_hv_sub()`.

9.3 collection.h File Reference

File containing I/O types and the `collection` class.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <map>
#include <set>
#include <sstream>
#include <o2scl/misc.h>
#include <o2scl/err_hnd.h>
#include <o2scl/test_mgr.h>
#include <o2scl/text_file.h>
#include <o2scl/file_format.h>
#include <o2scl/file_detect.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/omatrix_tlate.h>
#include <o2scl/ovector_cx_tlate.h>
#include <o2scl/omatrix_cx_tlate.h>
```

9.3.1 Detailed Description

File containing I/O types and the [collection](#) class.

New update todos:

- There are a lot of blank functions in the [io_base](#) class, so this class should be made abstract. This can actually be done by separating [io_base](#) into two classes, so that the parents of the [collection](#) class and the [io_tlate](#) class are different.
- Construct new templated add() function
- Fix test_type()
- Fix error handling in get() functions and make a lookup() method which doesn't necessarily call the error handler
- Fix documentation?
- Columnify summary output
- Ensure all parameters are matched by parameter and name if possible

Idea for future

Figure out what to do with templated matrix and vector output. What might be ideal is a "vector_io" type which will input or output a generic vector of any type.

Definition in file [collection.h](#).

Data Structures

- struct [collection_entry](#)

- struct `pointer_output`
A pointer output structure.
- struct `pointer_input`
A pointer input structure.
- class `io_base`
I/O base class.
- class `io_manager`
Manage I/O type information.
- class `io_type_info`
User interface to provide I/O type information.
- class `collection`
Collection of objects.
- class `collection::iterator`
An `iterator` for stepping through a `collection`.
- class `collection::type_iterator`
An `iterator` for stepping through the entries in a `collection` of a particular type.
- class `cinput`
Class to control object input.
- class `coutput`
Class to control object output.
- struct `coutput::ltptr`
Order the pointers by numeric value.
- class `io_vtlate`
A template for adding I/O classes.
- class `io_tlalte`
A template for adding I/O classes.
- class `bool_io_type`
I/O object for bool variables.
- class `char_io_type`
I/O object for char variables.
- class `double_io_type`
I/O object for double variables.
- class `int_io_type`
I/O object for int variables.
- class `long_io_type`
I/O object for long variables.
- class `string_io_type`
I/O object for string variables.
- class `word_io_type`
I/O object for words.

Functions

- int `operator==` (const `collection::iterator` &i1, const `collection::iterator` &i2)
Equality comparison for two iterators.
- int `operator!=` (const `collection::iterator` &i1, const `collection::iterator` &i2)
Inequality comparison for two iterators.
- int `operator==` (const `collection::type_iterator` &i1, const `collection::type_iterator` &i2)
Equality comparison for two iterators.
- int `operator!=` (const `collection::type_iterator` &i1, const `collection::type_iterator` &i2)
Inequality comparison for two iterators.

9.4 columnify.h File Reference

Functions to create output in columns.

```
#include <iostream>
#include <string>
#include <vector>
#include <o2scl/misc.h>
#include <o2scl/array.h>
```

9.4.1 Detailed Description

Functions to create output in columns.

Definition in file [columnify.h](#).

Data Structures

- class [columnify](#)
Create nicely formatted columns from a [table](#) of strings.

Functions

- `template<class mat_t >`
`int matrix_out_paren (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)`
A operator for simple matrix output using `operator()`.
- `template<class mat_t >`
`int matrix_cx_out_paren (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)`
A operator for simple complex matrix output using `operator()`.
- `template<class mat_t >`
`int matrix_out (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)`
A operator for simple matrix output using `operator[]`.

9.4.2 Function Documentation

9.4.2.1 `int matrix_out (std::ostream & os, mat_t & A, size_t nrows, size_t ncols)` [inline]

A operator for simple matrix output using `operator[]`.

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.

Idea for future

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 310 of file `columnify.h`.

9.4.2.2 int matrix_out_paren (std::ostream & os, mat_t & A, size_t nrows, size_t ncols) [inline]

A operator for simple matrix output using `operator()`.

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.

Definition at line 239 of file `columnify.h`.

9.5 cx_arith.h File Reference

Complex arithmetic.

```
#include <iostream>
#include <complex>
#include <cmath>
#include <gsl/gsl_math.h>
#include <gsl/gsl_complex.h>
#include <gsl/gsl_complex_math.h>
```

9.5.1 Detailed Description

Complex arithmetic.

Note:

One should be careful about including this header file, especially in other header files as it overloads some of the standard mathematical functions, i.e. `sqrt()`. If you need access to both these functions and the standard functions for double objects, the easiest way is probably to include this file in a source (not header file) and use `using namespace std`.

This used to be in a separate namespace, called `o2scl_arith`, but this causes problems with Koenig lookup in template classes for `operator*()` when defined for vector addition (for example).

Todo

Ensure all the functions are tested.

Idea for future

Define operators with assignment for complex + double?

Definition in file `cx_arith.h`.

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.

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`)

9.6 `err_hnd.h` File Reference

File for definitions for `err_base` and `err_class`.

```
#include <iostream>
#include <string>
#include <gsl/gsl_errno.h>
```

9.6.1 Detailed Description

File for definitions for `err_base` and `err_class`.

Definition in file `err_hnd.h`.

Data Structures

- class `err_base`
Class defining an error handler [abstract base].
- class `err_class`
The error handler.

Defines

- `#define O2SCL_ERR(d, n) o2scl::set_err_fn(d, __FILE__, __LINE__, n);`
Set an error with message `d` and code `n`.
- `#define O2SCL_CONV(d, n, b) { if (b) o2scl::set_err_fn(d, __FILE__, __LINE__, n); }`
Set a "convergence" error.

- `#define O2SCL_ERR2(d, d2, n)`
Set an error, two-string version.
- `#define O2SCL_CONV2(d, d2, n, b)`
Set a "convergence" error, two-string version.
- `#define O2SCL_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 O2SCL_CONV_RET(d, n, b)`
Set a "convergence" error and return the error value.
- `#define O2SCL_ERR2_RET(d, d2, n)`
Set an error and return the error value, two-string version.
- `#define O2SCL_CONV2_RET(d, d2, n, b)`
Set an error and return the error value, two-string version.
- `#define O2SCL_ASSERT(ev)`
A version of `assert`, i.e. exit if the error value is non-zero and do nothing otherwise.
- `#define O2SCL_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_ellos = 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_etolg = 30, gsl_etolg = 31, gsl_eof = 32, gsl_enotfound = 33,`
`gsl_ememtype = 34, gsl_eilenotfound = 35, gsl_eindex = 36, gsl_outsidecons = 37 }`
The integer error definitions.

Functions

- `void set_err_fn (const char *desc, const char *file, int line, int errnum)`
Set an error.
- `void error_update (int &ret, int err)`
Update an error value `err` with the value in `ret`.

Variables

- `err_base * err_hnd`
The global error handler pointer.
- `err_class def_err_hnd`
The default error handler.

9.6.2 Define Documentation

9.6.2.1 #define O2SCL_ASSERT(ev)

A version of `assert`, i.e. exit if the error value is non-zero and do nothing otherwise.

Idea for future

Make this consistent with `assert()` using `NDEBUG`?

Definition at line 354 of file `err_hnd.h`.

9.6.2.2 #define O2SCL_CONV_RET(d, n, b)

Value:

```
do { if (!b) { return 0; } else {
    o2scl::set_err_fn(d, __FILE__, __LINE__, n); return n; } } while (0) \
```

Set a "convergence" error and return the error value.

Definition at line 317 of file `err_hnd.h`.

9.6.2.3 #define O2SCL_ERR2(d, d2, n)

Value:

```
o2scl::set_err_fn((std::string(d)+d2).c_str(), \
    __FILE__, __LINE__, n);
```

Set an error, two-string version.

Definition at line 301 of file `err_hnd.h`.

9.6.2.4 #define O2SCL_ERR2_RET(d, d2, n)

Value:

```
do { o2scl::set_err_fn((std::string(d)+d2).c_str(), \
    __FILE__, __LINE__, n); return n; } while (0)
```

Set an error and return the error value, two-string version.

Definition at line 323 of file `err_hnd.h`.

9.6.3 Enumeration Type Documentation

9.6.3.1 anonymous enum

The integer error definitions.

The errors associated with the integers between -2 and 32 are from GSL, the rest are specific to `O2scl`.

Enumerator:

gsl_success Success (GSL).

gsl_failure Failure (GSL).

gsl_continue iteration has not converged (GSL)

gsl_edom input domain error, e.g. `sqrt(-1)` (GSL)

gsl_erange output range error, e.g. `exp(1e100)` (GSL)

gsl_efault invalid pointer (GSL)

gsl_einval invalid argument supplied by user (GSL)

gsl_efailed generic failure (GSL)

gsl_efactor factorization failed (GSL)

gsl_esanity sanity check failed - shouldn't happen (GSL)

gsl_enomem malloc failed (GSL)

gsl_ebadfunc problem with user-supplied function (GSL)

gsl_erunaway iterative process is out of control (GSL)

gsl_emaxiter exceeded max number of iterations (GSL)

gsl_ezerodiv tried to divide by zero (GSL)

gsl_ebadtol user specified an invalid tolerance (GSL)

gsl_etol failed to reach the specified tolerance (GSL)

gsl_eundrflw underflow (GSL)

gsl_eovrflw overflow (GSL)

gsl_eloss loss of accuracy (GSL)

gsl_eround failed because of roundoff error (GSL)

gsl_ebadlen matrix, vector lengths are not conformant (GSL)

gsl_enotsqr matrix not square (GSL)

gsl_esing apparent singularity detected (GSL)

gsl_ediverge integral or series is divergent (GSL)

gsl_eunsup requested feature is not supported by the hardware (GSL)

gsl_eunimpl requested feature not (yet) implemented (GSL)

gsl_ecache cache limit exceeded (GSL)

gsl_etable [table](#) limit exceeded (GSL)

gsl_enoprog iteration is not making progress toward solution (GSL)

gsl_enoprogi [jacobian](#) evaluations are not improving the solution (GSL)

gsl_etolf cannot reach the specified tolerance in `f` (GSL)

gsl_etolx cannot reach the specified tolerance in `x` (GSL)

gsl_etolg cannot reach the specified tolerance in [gradient](#) (GSL)

gsl_eof end of file (GSL)

gsl_enotfound Generic "not found" result.

gsl_ememtype Incorrect type for memory object.

gsl_efilenotfound File not found.

gsl_eindex Invalid index for array or matrix.

gsl_outsidecons Outside constraint region.

Definition at line 44 of file `err_hnd.h`.

9.6.4 Function Documentation

9.6.4.1 void error_update (int &ret, int err) [inline]

Update an error value `err` with the value in `ret`.

If `ret` is zero, this sets `ret` to the value `err`, and if `ret` is nonzero this function does nothing.

Definition at line 347 of file `err_hnd.h`.

9.7 givens.h File Reference

File for Givens rotations.

```
#include <gsl/gsl_math.h>
#include <o2scl/err_hnd.h>
#include <o2scl/permutation.h>
#include <o2scl/cblas.h>
#include <o2scl/vec_arith.h>
#include <o2scl/givens_base.h>
```

9.7.1 Detailed Description

File for Givens rotations.

Definition in file [givens.h](#).

Namespaces

- namespace [o2scl_linalg](#)
The namespace for linear algebra classes and functions.

Defines

- `#define O2SCL_IX(V, i) V[i]`
- `#define O2SCL_IX2(M, i, j) M[i][j]`

Functions

- void [create_givens](#) (const double a, const double b, double &c, double &s)
Desc.

9.8 givens_base.h File Reference

File for Givens rotations.

9.8.1 Detailed Description

File for Givens rotations.

Definition in file [givens_base.h](#).

Namespaces

- namespace [o2scl_linalg](#)
The namespace for linear algebra classes and functions.

Functions

- template<class mat1_t, class mat2_t >
void [apply_givens_qr](#) (size_t M, size_t N, mat1_t &Q, mat2_t &R, size_t i, size_t j, double c, double s)
Apply a rotation to matrices from the QR decomposition.
- template<class mat1_t, class mat2_t >
void [apply_givens_lq](#) (size_t M, size_t N, mat1_t &Q, mat2_t &L, size_t i, size_t j, double c, double s)
Apply a rotation to matrices from the LQ decomposition.
- template<class vec_t >
void [apply_givens_vec](#) (vec_t &v, size_t i, size_t j, double c, double s)
Apply a rotation to a vector; $v \rightarrow G^T v$.

9.9 hh_base.h File Reference

File for householder solver.

```
#include <o2scl/err_hnd.h>
#include <o2scl/cblas.h>
#include <o2scl/permutation.h>
#include <o2scl/vec_arith.h>
```

9.9.1 Detailed Description

File for householder solver.

Definition in file [hh_base.h](#).

Namespaces

- namespace [o2scl_linalg](#)
The namespace for linear algebra classes and functions.

Functions

- template<class mat_t, class vec_t >
int [HH_solve](#) (size_t n, mat_t &A, const vec_t &b, vec_t &x)
Solve linear system after Householder decomposition.
- template<class mat_t, class vec_t >
int [HH_svx](#) (size_t N, size_t M, mat_t &A, vec_t &x)
Solve a linear system after Householder decomposition in place.

9.10 householder_base.h File Reference

File for Householder transformations.

```
#include <o2scl/err_hnd.h>
```

```
#include <o2scl/cblas.h>
#include <o2scl/permutation.h>
#include <o2scl/vec_arith.h>
```

9.10.1 Detailed Description

File for Householder transformations.

Definition in file [householder_base.h](#).

Namespaces

- namespace [o2scl_linalg](#)
The namespace for linear algebra classes and functions.

Functions

- template<class vec_t >
double [householder_transform](#) (const size_t n, vec_t &v)
Replace the vector v with a householder vector and a coefficient tau that annihilates the last $n-1$ elements of v .
- template<class mat_t >
double [householder_transform_subcol](#) (mat_t &A, const size_t ir, const size_t ic, const size_t n)
Compute the householder transform of a vector formed with the last n rows of a column of a matrix.
- template<class vec_t, class mat_t >
int [householder_hm](#) (const size_t M, const size_t N, double tau, const vec_t &v, mat_t &A)
Apply a householder transformation v, τ to matrix m .
- template<class mat_t >
int [householder_hm_sub](#) (mat_t &M, const size_t ir, const size_t ic, const size_t nr, const size_t nc, const mat_t &M2, const size_t ir2, const size_t ic2, double tau)
Apply a householder transformation v, τ to submatrix of m .
- template<class vec_t >
int [householder_hv](#) (const size_t N, double tau, const vec_t &v, vec_t &w)
Apply a householder transformation v to vector w .
- template<class mat_t, class vec_t >
int [householder_hv_sub](#) (const mat_t &M, vec_t &w, double tau, const size_t ie, const size_t N)
Apply a householder transformation v to vector w .
- template<class mat1_t, class mat2_t >
int [householder_hm_sub2](#) (const size_t M, const size_t ic, double tau, const mat1_t &mv, mat2_t &A)
Special version of householder transformation for [QR_unpack\(\)](#).

9.11 lib_settings.h File Reference

File for definitions for [lib_settings_class](#).

```
#include <iostream>
#include <string>
```

9.11.1 Detailed Description

File for definitions for [lib_settings_class](#).

Definition in file [lib_settings.h](#).

Data Structures

- class [lib_settings_class](#)
A class to manage global library settings.

Namespaces

- namespace [o2scl](#)
The main O₂scl namespace.
- namespace [o2scl_linalg](#)
The namespace for linear algebra classes and functions.
- namespace [o2scl_linalg_paren](#)
The namespace for linear algebra classes and functions with operator().

Variables

- [lib_settings_class](#) [lib_settings](#)
The global library settings object.

9.11.2 Variable Documentation

9.11.2.1 lib_settings_class lib_settings

The global library settings object.

This global object is used by [polylog](#) and some of the O₂scl_eos classes to find data files. It may also be used by the end-user to probe details of the O₂scl installation.

9.12 lu_base.h File Reference

File for LU decomposition and associated solver.

9.12.1 Detailed Description

File for LU decomposition and associated solver.

Definition in file [lu_base.h](#).

Namespaces

- namespace [o2scl_linalg](#)
The namespace for linear algebra classes and functions.

Functions

- template<class mat_t >
int [LU_decomp](#) (const size_t N, mat_t &A, o2scl::permutation &p, int &signum)
Compute the LU decomposition of the matrix A.
- template<class mat_t, class vec_t >
int [LU_solve](#) (const size_t N, const mat_t &LU, const o2scl::permutation &p, const vec_t &b, vec_t &x)
Solve a linear system after LU decomposition.
- template<class mat_t, class vec_t >
int [LU_svx](#) (const size_t N, const mat_t &LU, const o2scl::permutation &p, vec_t &x)

Solve a linear system after LU decomposition in place.

- template<class mat_t, class vec_t >
int [LU_refine](#) (const size_t N, const mat_t &A, const mat_t &LU, const o2scl::permutation &p, const vec_t &b, vec_t &x, vec_t &residual)

Refine the solution of a linear system.

- template<class mat_t, class mat_col_t >
int [LU_invert](#) (const size_t N, const mat_t &LU, const o2scl::permutation &p, mat_t &inverse)

Compute the inverse of a matrix from its LU decomposition.

- template<class mat_t >
double [LU_det](#) (const size_t N, const mat_t &LU, int signum)

Compute the determinant of a matrix from its LU decomposition.

- template<class mat_t >
double [LU_lndet](#) (const size_t N, const mat_t &LU)

Compute the logarithm of the absolute value of the determinant of a matrix from its LU decomposition.

- template<class mat_t >
int [LU_sgndet](#) (const size_t N, const mat_t &LU, int signum)

Compute the sign of the determinant of a matrix from its LU decomposition.

9.13 minimize.h File Reference

One-dimensional minimization routines.

```
#include <o2scl/err_hnd.h>
```

9.13.1 Detailed Description

One-dimensional minimization routines.

Definition in file [minimize.h](#).

Data Structures

- class [minimize](#)
One-dimensional minimization [abstract base].
- class [minimize_bkt](#)
One-dimensional bracketing minimization [abstract base].
- class [minimize_de](#)
One-dimensional minimization using derivatives [abstract 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.

9.13.2 Function Documentation

9.13.2.1 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 near $x = c - w$ or $x = c + w$ is h (the value of the function exactly at these points is zero) 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 may ignore this constraint.

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

9.13.2.2 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

$$h \left(\frac{x - c}{w} \right)^2 \left[1 + e^{t(x-c+w)(c+w-x)/w^2} \right]^{-1}$$

This function is continuous and differentiable. Note that if $x = c$, then the function returns zero.

The exponential is handled gracefully by assuming that anything smaller than $\exp(-\ell)$ is zero. This creates a small discontinuity which can be removed with the sufficiently large value of ℓ .

It is important to note that, for large distances of x from `center`, this scales quadratically. If you are trying to constrain a function which decreases faster than quadratically by making x far from `center`, then a minimizer may ignore this constraint.

In the limit $t \rightarrow \infty$, this function converges towards the squared value of `constraint()`, except exactly at the points $x = c - w$ and $x = c + w$.

Definition at line 419 of file `minimize.h`.

9.13.2.3 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 $h(c - x + w)/(w + w \exp(t(x - c)/w))$ and has the advantage of being a continuous and differentiable function. The value of the function exactly at $x = c$ is $h/2$, but for x just below c the function is h and just above c the function is quite small.

The exponential is handled gracefully by assuming that anything smaller than $\exp(-\ell)$ is zero. This creates a small discontinuity which can be removed with the sufficiently large value of ℓ .

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 may ignore this constraint.

In the limit $t \rightarrow \infty$, this function converges towards `lower_bound()`, except exactly at the point $x = c$.

Definition at line 482 of file `minimize.h`.

9.13.2.4 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 may ignore this constraint.

Definition at line 447 of file `minimize.h`.

9.14 misc.h File Reference

Miscellaneous functions.

```
#include <cstdlib>
#include <iostream>
#include <cmath>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <gsl/gsl_ieee_utils.h>
#include <o2scl/err_hnd.h>
#include <o2scl/lib_settings.h>
```

9.14.1 Detailed Description

Miscellaneous functions.

Definition in file [misc.h](#).

Data Structures

- struct [string_comp](#)
Simple 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.
- template<class string_arr_t>
int [screenify](#) (size_t nin, const string_arr_t &in_cols, std::vector< std::string > &out_cols, size_t 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.
- int [remove_whitespace](#) (std::string &s)
Desc.
- std::string [binary_to_hex](#) (std::string s)
Take a string of binary quads and compress them to hexadecimal digits.
- template<class type_t>
int [gsl_alloc_arrays](#) (size_t nv, size_t msize, const char *names[], type_t *ptrs[], std::string func_name)
A convenient function to allocate several arrays of the same size.

9.14.2 Function Documentation

9.14.2.1 `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.

9.14.2.2 `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.

9.14.2.3 `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=40` 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.

9.14.2.4 `int gsl_alloc_arrays (size_t nv, size_t msize, const char * names[], type_t * ptrs[], std::string func_name)` [inline]

A convenient function to allocate several arrays of the same size.

Allocate memory for `nv` vectors of size `msize` with names specified in `names` and function name `func_name` to produce pointers in `ptrs`. This function is used internally to allocate several GSL vectors in succession, taking care to call the error handler when necessary.

Idea for future

Fix so that `__FILE__` and `__LINE__` are implemented here.

Definition at line 279 of file `misc.h`.

9.14.2.5 `int screenify (size_t nin, const string_arr_t & in_cols, std::vector< std::string > & out_cols, size_t max_size = 80)` [inline]

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 `out_cols`.

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
```

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`.

Definition at line 95 of file `misc.h`.

9.15 `omatrix_cx_tlate.h` File Reference

File for definitions of complex matrices.

```
#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>
```

9.15.1 Detailed Description

File for definitions of complex matrices.

Definition in file `omatrix_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

9.16 `omatrix_tlate.h` File Reference

File for definitions of matrices.

```
#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>
#include <o2scl/array.h>
```

9.16.1 Detailed Description

File for definitions of matrices.

Idea for future

The `xmatrix` class demonstrates how `operator[]` could return an `ovector_array` object and thus provide more bounds-checking. This would demand including a new parameter in `omatrix_view_tlate` which contains the vector type.

Definition in file `omatrix_tlate.h`.

Data Structures

- class `omatrix_const_view_tlate`
A const matrix view of omatrix objects.
- class `omatrix_base_tlate`
A base class for omatrix and omatrix_view.

- class `omatrix_view_tlate`
A matrix view of double-precision numbers.
- class `omatrix_tlate`
A matrix of double-precision numbers.
- class `omatrix_array_tlate`
Create a matrix from an array.
- 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_const_diag_tlate`
Create a vector from the main diagonal.
- class `omatrix_alloc`
A simple class to provide an `allocate()` function for `omatrix`.

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_const_view_tlate`< double, gsl_matrix, gsl_block > `omatrix_const_view`
omatrix_const_view typedef
- typedef `omatrix_base_tlate`< double, gsl_matrix, gsl_block > `omatrix_base`
omatrix_base 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_const_diag_tlate`< double, gsl_matrix, gsl_vector, gsl_block > `omatrix_const_diag`
omatrix_const_diag typedef
- typedef `omatrix_array_tlate`< double, gsl_matrix, gsl_block > `omatrix_array`
omatrix_array 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_base_tlate`< int, gsl_matrix_int, gsl_block_int > `omatrix_int_base`
omatrix_int_base typedef
- typedef `omatrix_const_view_tlate`< int, gsl_matrix_int, gsl_block_int > `omatrix_int_const_view`
omatrix_int_const_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
- typedef [omatrix_const_diag_tlate](#)< int, gsl_matrix_int, gsl_vector_int, gsl_block_int > [omatrix_int_const_diag](#)
omatrix_int_const_diag typedef
- typedef [omatrix_array_tlate](#)< int, gsl_matrix_int, gsl_block_int > [omatrix_int_array](#)
omatrix_int_array typedef

Functions

- template<class data_t, class parent_t, class block_t >
std::ostream & [operator<<](#) (std::ostream &os, const [omatrix_const_view_tlate](#)< data_t, parent_t, block_t > &v)
A operator for output of omatrix objects.

9.16.2 Function Documentation

9.16.2.1 std::ostream& operator<< (std::ostream &os, const omatrix_const_view_tlate< data_t, parent_t, block_t > &v) [inline]

A operator for output of omatrix objects.

This outputs all of the matrix elements. Each row is output with an endl 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.

Idea for future

This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

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

9.17 ovector_cx_tlate.h File Reference

File for definitions of complex vectors.

```
#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>
```

9.17.1 Detailed Description

File for definitions of complex vectors.

Definition in file [ovector_cx_tlate.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 [ofvector_cx](#)
A complex 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

- 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.

9.17.2 Function Documentation

9.17.2.1 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 1079 of file ovector_cx_tlate.h.

9.18 ovector_rev_tlate.h File Reference

File for definitions of reversed vectors.

```
#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/ovector_tlate.h>
#include <o2scl/array.h>
#include <o2scl/vector.h>
```

9.18.1 Detailed Description

File for definitions of reversed vectors.

Definition in file [ovector_rev_tlate.h](#).

Data Structures

- 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.

Typedefs

- 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_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

9.19 ovector_tlate.h File Reference

File for definitions of vectors.

```
#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/uevector_tlate.h>
#include <o2scl/array.h>
```

```
#include <o2scl/vector.h>
```

9.19.1 Detailed Description

File for definitions of vectors.

Idea for future

Clean up maybe by moving, for example, ovector reverse classes to a different header file

Definition in file [ovector_tlate.h](#).

Data Structures

- class [ovector_const_view_tlate](#)
A const vector view with finite stride.
- class [ovector_base_tlate](#)
A base class for ovector and ovector_view.
- 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_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 [ofvector](#)
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_base_tlate](#)< double, gsl_vector, gsl_block > [ovector_base](#)
ovector_base typedef
- typedef [ovector_view_tlate](#)< double, gsl_vector, gsl_block > [ovector_view](#)
ovector_view typedef
- typedef [ovector_const_view_tlate](#)< double, gsl_vector, gsl_block > [ovector_const_view](#)
ovector_const_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_tlate](#)< int, gsl_vector_int, gsl_block_int > [ovector_int](#)
ovector_int typedef
- typedef [ovector_base_tlate](#)< int, gsl_vector_int, gsl_block_int > [ovector_int_base](#)
ovector_int_base typedef
- typedef [ovector_view_tlate](#)< int, gsl_vector_int, gsl_block_int > [ovector_int_view](#)
ovector_int_view typedef
- typedef [ovector_const_view_tlate](#)< int, gsl_vector_int, gsl_block_int > [ovector_int_const_base](#)
ovector_int_const_base 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

Functions

- template<class data_t, class vparent_t, class block_t >
std::ostream & [operator<<](#) (std::ostream &os, const [ovector_const_view_tlate](#)< data_t, vparent_t, block_t > &v)
A operator for naive vector output.

9.19.2 Function Documentation

9.19.2.1 `std::ostream& operator<< (std::ostream & os, const ovector_const_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. If the vector is empty, nothing is done.

Definition at line 1672 of file `ovector_tlate.h`.

9.20 permutation.h File Reference

File containing [permutation](#) class and associated functions.

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
```

```
#include <vector>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_permutation.h>
#include <o2scl/err_hnd.h>
#include <o2scl/string_conv.h>
#include <o2scl/uvector_tlate.h>
#include <o2scl/array.h>
#include <o2scl/vector.h>
```

9.20.1 Detailed Description

File containing [permutation](#) class and associated functions.

Definition in file [permutation.h](#).

Data Structures

- class [permutation](#)
A [permutation](#).

Functions

- `std::ostream & operator<< (std::ostream &os, const permutation &p)`
Output operator for permutations.

9.20.2 Function Documentation

9.20.2.1 `std::ostream& operator<< (std::ostream &os, const permutation &p)`

Output operator for permutations.

A space is output between the [permutation](#) elements but no space or newline character is output after the last element.

If the size is zero, this function outputs nothing and does not call the error handler.

9.21 poly.h File Reference

Classes for solving polynomials.

```
#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>
```

9.21.1 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`

Idea for future

The quartics are tested only for `a4=1`, which should probably be generalized.

Definition in file [poly.h](#).

Data Structures

- class [quadratic_real](#)
Solve a quadratic polynomial with real coefficients and real roots [abstract base].
- class [quadratic_real_coeff](#)
Solve a quadratic polynomial with real coefficients and complex roots [abstract base].
- class [quadratic_complex](#)
Solve a quadratic polynomial with complex coefficients and complex roots [abstract base].
- class [cubic_real](#)
Solve a cubic polynomial with real coefficients and real roots [abstract base].
- class [cubic_real_coeff](#)
Solve a cubic polynomial with real coefficients and complex roots [abstract base].
- class [cubic_complex](#)
Solve a cubic polynomial with complex coefficients and complex roots [abstract base].
- class [quartic_real](#)
Solve a quartic polynomial with real coefficients and real roots [abstract base].
- class [quartic_real_coeff](#)
Solve a quartic polynomial with real coefficients and complex roots [abstract base].
- class [quartic_complex](#)
Solve a quartic polynomial with complex coefficients and complex roots [abstract base].
- class [poly_real_coeff](#)
Solve a general polynomial with real coefficients and complex roots [abstract base].
- class [poly_complex](#)
Solve a general polynomial with complex coefficients [abstract base].
- 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 [simple_quartic_real](#)
Solve a quartic with real coefficients and real roots.
- class [simple_quartic_complex](#)
Solve a quartic with complex coefficients and complex roots.

9.22 qr_base.h File Reference

File for QR decomposition and associated solver.

```
#include <o2scl/householder.h>
```

```
#include <o2scl/givens.h>
```

9.22.1 Detailed Description

File for QR decomposition and associated solver.

Definition in file [qr_base.h](#).

Namespaces

- namespace [o2scl_linalg](#)
The namespace for linear algebra classes and functions.

Functions

- template<class mat_t, class vec_t >
int [QR_decomp](#) (size_t M, size_t N, mat_t &A, vec_t &tau)
Compute the QR decomposition of matrix A.
- template<class mat_t, class vec_t, class vec2_t >
int [QR_solve](#) (size_t N, const mat_t &QR, const vec_t &tau, const vec2_t &b, vec2_t &x)
Solve the system $Ax = b$ using the QR factorization.
- template<class mat_t, class vec_t, class vec2_t >
int [QR_svx](#) (size_t M, size_t N, const mat_t &QR, const vec_t &tau, vec2_t &x)
Solve the system $Ax = b$ in place using the QR factorization.
- template<class mat_t, class vec_t, class vec2_t >
int [QR_QTvec](#) (const size_t M, const size_t N, const mat_t &QR, const vec_t &tau, vec2_t &v)
Form the product $Q^T v$ from a QR factorized matrix.
- template<class mat1_t, class mat2_t, class mat3_t, class vec_t >
int [QR_unpack](#) (const size_t M, const size_t N, const mat1_t &QR, const vec_t &tau, mat2_t &Q, mat3_t &R)
Unpack the QR matrix to the individual Q and R components.
- template<class mat1_t, class mat2_t, class vec1_t, class vec2_t >
int [QR_update](#) (size_t M, size_t N, mat1_t &Q, mat2_t &R, vec1_t &w, vec2_t &v)
Update a QR factorisation for $A = QR$, $A' = A + u v^T$.

9.23 string_conv.h File Reference

Various string conversion functions.

```
#include <iostream>
```

```
#include <cmath>
```

```
#include <string>
```

```
#include <fstream>
#include <sstream>
#include <o2scl/lib_settings.h>
#include <gsl/gsl_ieee_utils.h>
```

9.23.1 Detailed Description

Various string conversion functions.

Definition in file [string_conv.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 btos` (bool b)
Convert a boolean value 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.
- `bool stob` (std::string s)
Convert a string to a boolean value.
- `double stod` (std::string s)
Convert a string to a double.
- `std::string double_to_ieee_string` (const double *x)
Convert a double to a string containing IEEE representation.
- `bool has_minus_sign` (double *x)
Find out if the number pointed to by x has a minus sign.

9.23.2 Function Documentation

9.23.2.1 `std::string btos` (bool b)

Convert a boolean value to a string.

This returns "1" for true and "0" for false.

9.23.2.2 `std::string double_to_ieee_string` (const 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 *.

9.23.2.3 `std::string dtos` (double x, int prec = 6, bool auto_prec = false)

Convert a double to a string.

If `auto_prec` is false, then the number is converted to a string in the `ios::scientific` mode, otherwise, neither the scientific or fixed mode flags are set and the number is converted to a string in "automatic" mode.

9.23.2.4 bool has_minus_sign (double *x)

Find out if the number pointed to by `x` has a minus sign.

This function returns true if the number pointed to by `x` has a minus sign using the GSL IEEE functions. It is useful, for example, in distinguishing "-0.0" from "+0.0".

9.23.2.5 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.

9.23.2.6 size_t size_of_exponent (double x)

Returns the number of characters required to display the exponent of `x` in scientific mode.

This returns 2 or 3, depending on whether or not the absolute magnitude of the exponent is greater than 100. It uses `stringstream` to convert the number to a string and counts the number of characters directly.

9.23.2.7 bool stob (std::string s)

Convert a string to a boolean value.

This returns true only if the string has at least one character and the first non-whitespace character is either `t`, `T`, or one of the numbers 1 through 9.

This function never fails (it just returns false for an empty string).

9.23.2.8 double stod (std::string s)

Convert a string to a double.

If this function fails it will call [O2SCL_ERR\(\)](#) and return zero.

9.23.2.9 int stoi (std::string s)

Convert a string to an integer.

If this function fails it will call [O2SCL_ERR\(\)](#) and return zero.

9.24 tensor.h File Reference

File for definitions of tensors.

```
#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>
```

```
#include <o2scl/umatrix_tlate.h>
#include <o2scl/smart_interp.h>
```

9.24.1 Detailed Description

File for definitions of tensors.

Definition in file [tensor.h](#).

Data Structures

- class [tensor](#)
Tensor class with arbitrary dimensions.
- class [tensor_grid](#)
Tensor class with arbitrary dimensions with a grid.
- class [tensor1](#)
Rank 1 [tensor](#).
- class [tensor2](#)
Rank 2 [tensor](#).
- class [tensor_grid2](#)
Rank 2 [tensor](#) with a grid.
- class [tensor3](#)
Rank 3 [tensor](#).
- class [tensor_grid3](#)
Rank 3 [tensor](#) with a grid.
- class [tensor4](#)
Rank 4 [tensor](#).
- class [tensor_grid4](#)
Rank 4 [tensor](#) with a grid.

9.25 tridiag_base.h File Reference

File for solving tridiagonal systems.

9.25.1 Detailed Description

File for solving tridiagonal systems.

Definition in file [tridiag_base.h](#).

Namespaces

- namespace [o2scl_linalg](#)
The namespace for linear algebra classes and functions.

Functions

- template<class vec_t, class vec2_t >
int [solve_tridiag_sym](#) (const vec_t &diag, const vec2_t &offdiag, const vec_t &b, vec_t &x, size_t N)
Solve a symmetric tridiagonal linear system.
- template<class vec_t, class vec2_t >
int [solve_tridiag_nonsym](#) (const vec_t &diag, const vec2_t &abovediag, const vec2_t &belowdiag, const vec_t &rhs, vec_t &x, size_t N)

Solve an asymmetric tridiagonal linear system.

- template<class vec_t >
int [solve_cyc_tridiag_sym](#) (const vec_t &diag, const vec_t &offdiag, const vec_t &b, vec_t &x, size_t N)
Solve a symmetric cyclic tridiagonal linear system.
- template<class vec_t >
int [solve_cyc_tridiag_nonsym](#) (const vec_t &diag, const vec_t &abovediag, const vec_t &belowdiag, const vec_t &rhs, vec_t &x, size_t N)
Solve an asymmetric cyclic tridiagonal linear system.

9.26 umatrix_cx_tlate.h File Reference

File for definitions of matrices.

```
#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>
#include <o2scl/uvector_cx_tlate.h>
```

9.26.1 Detailed Description

File for definitions of matrices.

Definition in file [umatrix_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.

9.26.2 Function Documentation

9.26.2.1 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 endl 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.

Note:

At the moment, this function assumes that scientific mode is on and showpos is off.

Idea for future

Make this function work even when scientific mode is not on, either by converting to scientific mode and converting back, or by leaving scientific mode off and padding with spaces

Definition at line 679 of file `umatrix_cx_tlate.h`.

9.27 umatrix_tlate.h File Reference

File for definitions of matrices.

```
#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>
```

9.27.1 Detailed Description

File for definitions of matrices.

Definition in file [umatrix_tlate.h](#).

Data Structures

- class [umatrix_const_view_tlate](#)
A matrix view of double-precision numbers.
- class [umatrix_base_tlate](#)
A matrix view of double-precision numbers.
- 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_base_tlate](#)< double > [umatrix_base](#)
umatrix_base typedef
- typedef [umatrix_const_view_tlate](#)< double > [umatrix_const_view](#)
umatrix_const_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_const_view_tlate](#)< int > [umatrix_int_const_view](#)
umatrix_int_const_view typedef
- typedef [umatrix_base_tlate](#)< int > [umatrix_int_base](#)
umatrix_int_base 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_const_view_tlate< data_t > &v)`
A operator for naive matrix output.

9.27.2 Function Documentation

9.27.2.1 std::ostream& operator<< (std::ostream & os, const umatrix_const_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 endl 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.

Idea for future

This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Definition at line 895 of file `umatrix_tlate.h`.

9.28 user_io.h File Reference

Support functions for I/O.

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

9.28.1 Detailed Description

Support functions for I/O.

Todo

Do input for array and 2d array objects, finish output functions. Make a macro for I/O for doubles, ints and other objects w/o a `type()` function.

Definition in file `user_io.h`.

Functions

- `template<class obj_t >`
`int o2scl_input(in_file_format *ins, obj_t *&obj, std::string &name, size_t &sz, size_t &sz2, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_input(in_file_format *ins, obj_t *&obj, std::string &name, size_t &sz, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_input(in_file_format *ins, obj_t *&obj, std::string &name, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_input_text(std::string fname, obj_t *&obj, std::string &name, size_t &sz, size_t &sz2, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_input_text(std::string fname, obj_t *&obj, std::string &name, size_t &sz, bool verbose=false)`
Input one object from a file.

- `template<class obj_t >`
`int o2scl_input_text (std::string fname, obj_t *&obj, std::string &name, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_input_name (in_file_format *ins, obj_t *&obj, std::string name, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_input_name_text (std::string fname, obj_t *&obj, std::string &name, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_input (in_file_format *ins, obj_t *&obj, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_input_text (std::string fname, obj_t *&obj, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_output (out_file_format *outs, obj_t *obj, std::string name, size_t &sz, size_t &sz2, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_output (out_file_format *outs, obj_t *obj, std::string name, size_t &sz, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_output (out_file_format *outs, obj_t *obj, std::string name, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_output_text (std::string fname, obj_t *obj, std::string name, size_t sz, size_t sz2, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_output_text (std::string fname, obj_t *obj, std::string name, size_t sz, bool verbose=false)`
Input one object from a file.
- `template<class obj_t >`
`int o2scl_output_text (std::string fname, obj_t *obj, std::string name, bool verbose=false)`
Input one object from a file.

9.28.2 Function Documentation

9.28.2.1 `int o2scl_input (in_file_format *ins, obj_t *&obj, bool verbose = false) [inline]`

Input one object from a file.

This requires a default copy constructor and a `type()` function.

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](#).

Definition at line 342 of file `user_io.h`.

9.28.2.2 `int o2scl_input (in_file_format *ins, obj_t *&obj, std::string &name, bool verbose = false) [inline]`

Input one object from a file.

This requires a default copy constructor and a `type()` function.

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](#).

Definition at line 165 of file `user_io.h`.

9.28.2.3 `int o2scl_input (in_file_format * ins, obj_t *& obj, std::string & name, size_t & sz, bool verbose = false) [inline]`

Input one object from a file.

This requires a default copy constructor and a type() function.

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](#).

Definition at line 109 of file user_io.h.

9.28.2.4 `int o2scl_input (in_file_format * ins, obj_t *& obj, std::string & name, size_t & sz, size_t & sz2, bool verbose = false) [inline]`

Input one object from a file.

This requires a default copy constructor and a type() function.

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](#).

Definition at line 50 of file user_io.h.

9.28.2.5 `int o2scl_input_name (in_file_format * ins, obj_t *& obj, std::string name, bool verbose = false) [inline]`

Input one object from a file.

This requires a default copy constructor and a type() function.

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](#).

Definition at line 270 of file user_io.h.

9.28.2.6 `int o2scl_input_name_text (std::string fname, obj_t *& obj, std::string & name, bool verbose = false) [inline]`

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](#).

Definition at line 324 of file user_io.h.

9.28.2.7 `int o2scl_input_text (std::string fname, obj_t *& obj, bool verbose = false) [inline]`

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](#).

Definition at line 396 of file user_io.h.

9.28.2.8 `int o2scl_input_text (std::string fname, obj_t *& obj, std::string & name, bool verbose = false) [inline]`

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](#).

Definition at line 252 of file user_io.h.

9.28.2.9 `int o2scl_input_text (std::string fname, obj_t * & obj, std::string & name, size_t & sz, bool verbose = false) [inline]`

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](#).

Definition at line 236 of file user_io.h.

9.28.2.10 `int o2scl_input_text (std::string fname, obj_t * & obj, std::string & name, size_t & sz, size_t & sz2, bool verbose = false) [inline]`

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](#).

Definition at line 219 of file user_io.h.

9.28.2.11 `int o2scl_output (out_file_format * outs, obj_t * obj, std::string name, bool verbose = false) [inline]`

Input one object from a file.

This requires a default copy constructor and a type() function.

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](#).

Todo

Do input for array and 2d array objects

Definition at line 467 of file user_io.h.

9.28.2.12 `int o2scl_output (out_file_format * outs, obj_t * obj, std::string name, size_t & sz, bool verbose = false) [inline]`

Input one object from a file.

This requires a default copy constructor and a type() function.

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](#).

Todo

Do input for array and 2d array objects

Definition at line 441 of file user_io.h.

9.28.2.13 `int o2scl_output (out_file_format * outs, obj_t * obj, std::string name, size_t & sz, size_t & sz2, bool verbose = false) [inline]`

Input one object from a file.

This requires a default copy constructor and a type() function.

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](#).

Todo

Do input for array and 2d array objects

Definition at line 416 of file `user_io.h`.

9.28.2.14 `int o2scl_output_text (std::string fname, obj_t * obj, std::string name, bool verbose = false) [inline]`

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](#).

Definition at line 520 of file `user_io.h`.

9.28.2.15 `int o2scl_output_text (std::string fname, obj_t * obj, std::string name, size_t sz, bool verbose = false) [inline]`

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](#).

Definition at line 503 of file `user_io.h`.

9.28.2.16 `int o2scl_output_text (std::string fname, obj_t * obj, std::string name, size_t sz, size_t sz2, bool verbose = false) [inline]`

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](#).

Definition at line 486 of file `user_io.h`.

9.29 `uvector_cx_tlate.h` File Reference

File for definitions of complex unit-stride vectors.

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <o2scl/err_hnd.h>
#include <gsl/gsl_vector.h>
```

9.29.1 Detailed Description

File for definitions of complex unit-stride vectors.

Definition in file `uvector_cx_tlate.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.

9.29.2 Function Documentation

9.29.2.1 `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 650 of file `uvector_cx_tlate.h`.

9.30 `uvector_tlate.h` File Reference

File for definitions of unit-stride vectors.

```
#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/array.h>
#include <o2scl/vector.h>
```

9.30.1 Detailed Description

File for definitions of unit-stride vectors.

Definition in file `uvector_tlate.h`.

Data Structures

- class `uvector_const_view_tlate`
A const vector view with unit stride.
- class `uvector_base_tlate`
Desc.
- class `uvector_view_tlate`
A base class for uvectors.
- 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 `uvector_alloc`
A simple class to provide an `allocate()` function for `uvector`.
- class `uvector_int_alloc`
A simple class to provide an `allocate()` function for `uvector_int`.
- class `ufvector`
A vector with unit-stride where the memory allocation is performed in the constructor.

Typedefs

- typedef `uvector_tlate< double > uvector`
uvector typedef
- typedef `uvector_view_tlate< double > uvector_view`
uvector_view typedef
- typedef `uvector_base_tlate< double > uvector_base`
uvector_base typedef
- typedef `uvector_const_view_tlate< double > uvector_const_view`
uvector_const_view typedef
- typedef `uvector_array_tlate< double > uvector_array`
uvector_array typedef

- typedef `uvector_subvector_tlate< double > uvector_subvector`
uvector_subvector typedef
- typedef `uvector_const_array_tlate< double > uvector_const_array`
uvector_const_array typedef
- typedef `uvector_const_subvector_tlate< double > 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_const_view_tlate< data_t > &v`)
A operator for naive vector output.

9.30.2 Function Documentation

9.30.2.1 std::ostream& operator<< (std::ostream & os, const uvector_const_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. If the vector is empty, nothing is done.

Definition at line 996 of file `uvector_tlate.h`.

9.31 vec_arith.h File Reference

Vector and matrix arithmetic.

```
#include <iostream>
#include <complex>
#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>
```

9.31.1 Detailed Description

Vector and matrix arithmetic.

By default, the following operators are defined:

```

ovector operator+(ovector_const_view &x, ovector_const_view &y);
ovector operator+(ovector_const_view &x, uvector_const_view &y);
ovector operator+(uvector_const_view &x, ovector_const_view &y);
uvector operator+(uvector_const_view &x, uvector_const_view &y);

ovector_cx operator+(ovector_cx_view &x, ovector_cx_view &y);
ovector_cx operator+(ovector_cx_view &x, uvector_cx_view &y);
ovector_cx operator+(uvector_cx_view &x, ovector_cx_view &y);
uvector_cx operator+(uvector_cx_view &x, uvector_cx_view &y);

ovector operator-(ovector_const_view &x, ovector_const_view &y);
ovector operator-(ovector_const_view &x, uvector_const_view &y);
ovector operator-(uvector_const_view &x, ovector_const_view &y);
uvector operator-(uvector_const_view &x, uvector_const_view &y);

ovector_cx operator-(ovector_cx_view &x, ovector_cx_view &y);
ovector_cx operator-(ovector_cx_view &x, uvector_cx_view &y);
ovector_cx operator-(uvector_cx_view &x, ovector_cx_view &y);
uvector_cx operator-(uvector_cx_view &x, uvector_cx_view &y);

ovector operator*(omatrix_const_view &x, ovector_const_view &y);
ovector operator*(omatrix_const_view &x, uvector_const_view &y);
ovector operator*(umatrix_const_view &x, ovector_const_view &y);
uvector operator*(umatrix_const_view &x, uvector_const_view &y);

ovector_cx operator*(omatrix_cx_view &x, ovector_cx_view &y);
ovector_cx operator*(omatrix_cx_view &x, uvector_cx_view &y);
ovector_cx operator*(umatrix_cx_view &x, ovector_cx_view &y);
uvector_cx operator*(umatrix_cx_view &x, uvector_cx_view &y);

ovector operator*(ovector_const_view &x, omatrix_const_view &y);
ovector operator*(ovector_const_view &x, umatrix_const_view &y);
ovector operator*(uvector_const_view &x, omatrix_const_view &y);
uvector operator*(uvector_const_view &x, umatrix_const_view &y);

ovector trans_mult(ovector_const_view &x, omatrix_const_view &y);
ovector trans_mult(ovector_const_view &x, umatrix_const_view &y);
ovector trans_mult(uvector_const_view &x, omatrix_const_view &y);
uvector trans_mult(uvector_const_view &x, umatrix_const_view &y);

double dot(ovector_const_view &x, ovector_const_view &y);
double dot(ovector_const_view &x, uvector_const_view &y);
double dot(uvector_const_view &x, ovector_const_view &y);
double dot(uvector_const_view &x, uvector_const_view &y);

double dot(ovector_cx_view &x, ovector_cx_view &y);
double dot(ovector_cx_view &x, uvector_cx_view &y);
double dot(uvector_cx_view &x, ovector_cx_view &y);
double dot(uvector_cx_view &x, uvector_cx_view &y);

ovector operator*(double x, ovector_const_view &y);
uvector operator*(double x, uvector_const_view &y);
ovector operator*(ovector_const_view &x, double y);
uvector operator*(uvector_const_view &x, double y);

ovector pair_prod(ovector_const_view &x, ovector_const_view &y);
ovector pair_prod(ovector_const_view &x, uvector_const_view &y);
ovector pair_prod(uvector_const_view &x, ovector_const_view &y);
uvector pair_prod(uvector_const_view &x, uvector_const_view &y);

ovector_cx pair_prod(ovector_cx_view &x, ovector_const_view &y);
ovector_cx pair_prod(ovector_cx_view &x, uvector_const_view &y);
ovector_cx pair_prod(uvector_cx_view &x, ovector_const_view &y);

```

```

uvector_cx pair_prod(uvector_cx_view &x, uvector_const_view &y);
ovector_cx pair_prod(ovector_const_view &x, ovector_cx_view &y);
ovector_cx pair_prod(ovector_const_view &x, uvector_cx_view &y);
ovector_cx pair_prod(uvector_const_view &x, ovector_cx_view &y);
uvector_cx pair_prod(uvector_const_view &x, uvector_cx_view &y);
ovector_cx pair_prod(ovector_cx_view &x, ovector_cx_view &y);
ovector_cx pair_prod(ovector_cx_view &x, uvector_cx_view &y);
ovector_cx pair_prod(uvector_cx_view &x, ovector_cx_view &y);
uvector_cx pair_prod(uvector_cx_view &x, uvector_cx_view &y);

bool operator==(ovector_const_view &x, ovector_const_view &y);
bool operator==(ovector_const_view &x, uvector_const_view &y);
bool operator==(uvector_const_view &x, ovector_const_view &y);
bool operator==(uvector_const_view &x, uvector_const_view &y);

bool operator!=(ovector_const_view &x, ovector_const_view &y);
bool operator!=(ovector_const_view &x, uvector_const_view &y);
bool operator!=(uvector_const_view &x, ovector_const_view &y);
bool operator!=(uvector_const_view &x, uvector_const_view &y);

```

Note:

This used to be in a separate namespace, called `o2scl_arith`, but this causes problems with Koenig lookup in template classes for `operator*()` when defined for vector addition (for example).

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`.

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)`
The header macro for vector-vector subtraction.
- `#define O2SCL_OP_MAT_VEC_MULT(vec1, vec2, mat)`
The header macro for matrix-vector (right) multiplication.
- `#define O2SCL_OP_CMAT_CVEC_MULT(vec1, vec2, mat)`
The header macro for complex matrix-vector (right) multiplication.
- `#define O2SCL_OP_VEC_MAT_MULT(vec1, vec2, mat)`
The header macro for vector-matrix (left) multiplication.
- `#define O2SCL_OP_TRANS_MULT(vec1, vec2, mat)`
*The header macro for the `trans_mult` form of vector * matrix.*
- `#define O2SCL_OP_DOT_PROD(dtype, vec1, vec2)`
The header macro for vector scalar (dot) product.
- `#define O2SCL_OP_CX_DOT_PROD(dtype, vec1, vec2)`
The header macro for complex vector scalar (dot) product.
- `#define O2SCL_OP_SCA_VEC_MULT(dtype, vecv, vec)`
The header macro for scalar-vector multiplication.
- `#define O2SCL_OP_VEC_SCA_MULT(dtype, vecv, vec)`
The header macro for vector-scalar multiplication.

- `#define O2SCL_OP_VEC_VEC_PRO(vec1, vec2, vec3)`
*The header macro for pairwise vector * vector (where either vector can be real or complex).*
- `#define O2SCL_OPSRC_VEC_VEC_ADD(vec1, vec2, vec3)`
The source code macro for vector-vector addition.
- `#define O2SCL_OPSRC_VEC_VEC_SUB(vec1, vec2, vec3)`
The source code macro for vector-vector subtraction.
- `#define O2SCL_OPSRC_MAT_VEC_MULT(vec1, vec2, mat)`
*The source code macro for matrix * vector.*
- `#define O2SCL_OPSRC_CMAT_CVEC_MULT(vec1, vec2, mat)`
*The source code macro for complex matrix * complex vector.*
- `#define O2SCL_OPSRC_VEC_MAT_MULT(vec1, vec2, mat)`
*The source code macro for the operator form of vector * matrix.*
- `#define O2SCL_OPSRC_TRANS_MULT(vec1, vec2, mat)`
*The source code macro for the trans_mult form of vector * matrix.*
- `#define O2SCL_OPSRC_DOT_PROD(dtype, vec1, vec2)`
The source code macro for a vector dot product.
- `#define O2SCL_OPSRC_CX_DOT_PROD(dtype, vec1, vec2)`
The source code macro for a complex vector dot product.
- `#define O2SCL_OPSRC_SCA_VEC_MULT(dtype, vecv, vec)`
*The source code macro for vector=scalar*vector.*
- `#define O2SCL_OPSRC_VEC_SCA_MULT(dtype, vecv, vec)`
*The source code macro for vector=vector*scalar.*
- `#define O2SCL_OPSRC_VEC_VEC_PRO(vec1, vec2, vec3)`
*The source code macro for pairwise vector * vector (where either vector can be real or complex).*
- `#define O2SCL_OP_VEC_VEC_EQUAL(vec1, vec2)`
The header macro for vector==vector.
- `#define O2SCL_OPSRC_VEC_VEC_EQUAL(vec1, vec2)`
The source code macro vector==vector.
- `#define O2SCL_OP_VEC_VEC_NEQUAL(vec1, vec2)`
The header macro for vector!=vector.
- `#define O2SCL_OPSRC_VEC_VEC_NEQUAL(vec1, vec2)`
The source code macro vector!=vector.

9.31.2 Define Documentation

9.31.2.1 #define O2SCL_OP_CMAT_CVEC_MULT(vec1, vec2, mat)

Value:

```
vec1 operator* \
    (const mat &m, const vec2 &x);
```

The header macro for complex matrix-vector (right) multiplication.

Given types `vec1`, `vec2`, and `mat`, this macro provides the function declaration for adding two vectors using the form

```
vec1 operator*(const mat &m, const vec3 &x);
```

The corresponding definition is given in `O2SCL_OPSRC_CMAT_CVEC_MULT`.

Definition at line 283 of file `vec_arith.h`.

9.31.2.2 #define O2SCL_OP_CX_DOT_PROD(dtype, vec1, vec2)

Value:

```
dtype dot \
    (const vec1 &x, const vec2 &y);
```

The header macro for complex vector scalar (dot) product.

Given types `vec1`, `vec2`, and `dtype`, this macro provides the function declaration for adding two vectors using the form

```
dtype operator*(const vec1 &x, const vec2 &y);
```

The corresponding definition is given in [O2SCL_OP_SRC_CX_DOT_PROD](#).

Definition at line 409 of file `vec_arith.h`.

9.31.2.3 #define O2SCL_OP_DOT_PROD(dtype, vec1, vec2)

Value:

```
dtype dot
(const vec1 &x, const vec2 &y);
```

The header macro for vector scalar (dot) product.

Given types `vec1`, `vec2`, and `dtype`, this macro provides the function declaration for adding two vectors using the form

```
dtype operator*(const vec1 &x, const vec2 &y);
```

The corresponding definition is given in [O2SCL_OP_SRC_DOT_PROD](#).

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

9.31.2.4 #define O2SCL_OP_MAT_VEC_MULT(vec1, vec2, mat)

Value:

```
vec1 operator*
(const mat &m, const vec2 &x);
```

The header macro for matrix-vector (right) multiplication.

Given types `vec1`, `vec2`, and `mat`, this macro provides the function declaration for adding two vectors using the form

```
vec1 operator*(const mat &m, const vec3 &x);
```

See also the blas version of this function [dgemv\(\)](#).

The corresponding definition is given in [O2SCL_OP_SRC_MAT_VEC_MULT](#).

Definition at line 247 of file `vec_arith.h`.

9.31.2.5 #define O2SCL_OP_SCA_VEC_MULT(dtype, vecv, vec)

Value:

```
vec operator*
(const dtype &x, const vecv &y);
```

The header macro for scalar-vector multiplication.

Given types `vecv`, `vec`, and `dtype`, this macro provides the function declaration for adding two vectors using the form

```
vec operator*(const dtype &x, const vecv &y);
```

The corresponding definition is given in [O2SCL_OP_SRC_SCA_VEC_MULT](#).

Definition at line 443 of file `vec_arith.h`.

9.31.2.6 #define O2SCL_OP_TRANS_MULT(vec1, vec2, mat)**Value:**

```
vec1 trans_mult \
    (const vec2 &x, const mat &m);
```

The header macro for the `trans_mult` form of vector * matrix.

Definition at line 340 of file `vec_arith.h`.

9.31.2.7 #define O2SCL_OP_VEC_MAT_MULT(vec1, vec2, mat)**Value:**

```
vec1 operator* \
    (const vec2 &x, const mat &m);
```

The header macro for vector-matrix (left) multiplication.

Given types `vec1`, `vec2`, and `mat`, this macro provides the function declaration for adding two vectors using the form

```
vec1 operator*(const vec3 &x, const mat &m);
```

The corresponding definition is given in [O2SCL_OP_SRC_VEC_MAT_MULT](#).

Definition at line 318 of file `vec_arith.h`.

9.31.2.8 #define O2SCL_OP_VEC_SCA_MULT(dtype, vecv, vec)**Value:**

```
vec operator* \
    (const vecv &x, const dtype &y);
```

The header macro for vector-scalar multiplication.

Given types `vecv`, `vec`, and `dtype`, this macro provides the function declaration for adding two vectors using the form

```
vec operator*(const vecv &x, const dtype &y);
```

The corresponding definition is given in [O2SCL_OP_SRC_VEC_SCA_MULT](#).

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

9.31.2.9 #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.

Given types `vec1`, `vec2`, and `vec_3`, this macro provides the function declaration for adding two vectors using the form

```
vec1 operator+(const vec2 &x, const vec3 &y);
```

The corresponding definition is given in [O2SCL_OP_SRC_VEC_VEC_ADD](#).

Note:

This used to be in a separate namespace, called `o2scl_arith`, but this causes problems with Koenig lookup in template classes for `operator*()` when defined for vector addition (for example).

Definition at line 160 of file `vec_arith.h`.

9.31.2.10 #define O2SCL_OP_VEC_VEC_EQUAL(vec1, vec2)

Value:

```
bool operator== \
    (const vec1 &x, const vec2 &y);
```

The header macro for `vector==vector`.

Given types `vec1` and `vec2`, this macro provides the function declaration for vector equality comparisons using

```
bool operator==(const vec1 &x, const vec2 &y);
```

Note:

Two vectors with different sizes are defined to be not equal, no matter what their contents.

The corresponding definition is given in [O2SCL_OP_SRC_VEC_VEC_EQUAL](#).

Definition at line 762 of file `vec_arith.h`.

9.31.2.11 #define O2SCL_OP_VEC_VEC_NEQUAL(vec1, vec2)

Value:

```
bool operator!= \
    (const vec1 &x, const vec2 &y);
```

The header macro for `vector!=vector`.

Given types `vec1` and `vec2`, this macro provides the function declaration for vector inequality comparisons using

```
bool operator==(const vec1 &x, const vec2 &y);
```

Note:

Two vectors with different sizes are defined to be not equal, no matter what their contents.

The corresponding definition is given in [O2SCL_OP_SRC_VEC_VEC_NEQUAL](#).

Definition at line 844 of file `vec_arith.h`.

9.31.2.12 #define O2SCL_OP_VEC_VEC_PROD(vec1, vec2, vec3)

Value:

```
vec1 pair_prod \
    (const vec2 &x, const vec3 &y);
```


The header macro for pairwise vector * vector (where either vector can be real or complex).

Given types `vec1`, `vec2`, and `vec3`, this macro provides the function declaration for adding two vectors using the form

```
vec1 pair_prod(const vec2 &x, const vec3 &y);
```

The corresponding definition is given in [O2SCL_OPSRC_VEC_VEC_PRO](#).

Definition at line 500 of file `vec_arith.h`.

9.31.2.13 #define O2SCL_OP_VEC_VEC_SUB(vec1, vec2, vec3)

Value:

```
vec1 operator- \
(const vec2 &x, const vec3 &y);
```

The header macro for vector-vector subtraction.

Given types `vec1`, `vec2`, and `vec3`, this macro provides the function declaration for adding two vectors using the form

```
vec1 operator-(const vec2 &x, const vec3 &y);
```

The corresponding definition is given in [O2SCL_OPSRC_VEC_VEC_SUB](#).

Definition at line 202 of file `vec_arith.h`.

9.31.2.14 #define O2SCL_OPSRC_CMAT_CVEC_MULT(vec1, vec2, mat)

The source code macro for complex matrix * complex vector.

This define macro generates the function definition. See the function declaration [O2SCL_OP_CMAT_CVEC_MULT](#)

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

9.31.2.15 #define O2SCL_OPSRC_CX_DOT_PROD(dtype, vec1, vec2)

The source code macro for a complex vector dot product.

This define macro generates the function definition. See the function declaration [O2SCL_OP_CX_DOT_PROD](#)

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

9.31.2.16 #define O2SCL_OPSRC_DOT_PROD(dtype, vec1, vec2)

The source code macro for a vector dot product.

This define macro generates the function definition. See the function declaration [O2SCL_OP_DOT_PROD](#)

Definition at line 669 of file `vec_arith.h`.

9.31.2.17 #define O2SCL_OPSRC_MAT_VEC_MULT(vec1, vec2, mat)

The source code macro for matrix * vector.

This define macro generates the function definition. See the function declaration [O2SCL_OP_MAT_VEC_MULT](#)

Definition at line 580 of file `vec_arith.h`.

9.31.2.18 #define O2SCL_OP_SRC_SCA_VEC_MULT(dtype, vecv, vec)

The source code macro for vector=scalar*vector.

This define macro generates the function definition. See the function declaration [O2SCL_OP_SRC_SCA_VEC_MULT](#)
Definition at line 703 of file vec_arith.h.

9.31.2.19 #define O2SCL_OP_SRC_TRANS_MULT(vec1, vec2, mat)

The source code macro for the trans_mult form of vector * matrix.

This define macro generates the function definition. See the function declaration [O2SCL_OP_SRC_TRANS_MULT](#)
Definition at line 648 of file vec_arith.h.

9.31.2.20 #define O2SCL_OP_SRC_VEC_MAT_MULT(vec1, vec2, mat)

The source code macro for the operator form of vector * matrix.

This define macro generates the function definition. See the function declaration [O2SCL_OP_SRC_VEC_MAT_MULT](#)
Definition at line 626 of file vec_arith.h.

9.31.2.21 #define O2SCL_OP_SRC_VEC_SCA_MULT(dtype, vecv, vec)

The source code macro for vector=vector*scalar.

This define macro generates the function definition. See the function declaration [O2SCL_OP_SRC_VEC_SCA_MULT](#)
Definition at line 719 of file vec_arith.h.

9.31.2.22 #define O2SCL_OP_SRC_VEC_VEC_ADD(vec1, vec2, vec3)

The source code macro for vector-vector addition.

This define macro generates the function definition. See the function declaration [O2SCL_OP_SRC_VEC_VEC_ADD](#)
Definition at line 546 of file vec_arith.h.

9.31.2.23 #define O2SCL_OP_SRC_VEC_VEC_EQUAL(vec1, vec2)

The source code macro vector==vector.

Note:

Two vectors with different sizes are defined to be not equal, no matter what their contents.

This define macro generates the function definition. See the function declaration [O2SCL_OP_SRC_VEC_VEC_EQUAL](#)
Definition at line 818 of file vec_arith.h.

9.31.2.24 #define O2SCL_OP_SRC_VEC_VEC_NEQUAL(vec1, vec2)

The source code macro vector!=vector.

Note:

Two vectors with different sizes are defined to be not equal, no matter what their contents.

This define macro generates the function definition. See the function declaration [O2SCL_OP_SRC_VEC_VEC_NEQUAL](#)
Definition at line 900 of file vec_arith.h.

9.31.2.25 #define O2SCL_OP_SRC_VEC_VEC_PRO(vec1, vec2, vec3)

The source code macro for pairwise vector * vector (where either vector can be real or complex).

This define macro generates the function definition. See the function declaration [O2SCL_OP_VEC_VEC_PRO](#)

Definition at line 736 of file vec_arith.h.

9.31.2.26 #define O2SCL_OP_SRC_VEC_VEC_SUB(vec1, vec2, vec3)

The source code macro for vector-vector subtraction.

This define macro generates the function definition. See the function declaration [O2SCL_OP_VEC_VEC_SUB](#)

Definition at line 563 of file vec_arith.h.

9.32 vec_stats.h File Reference

File containing statistics template functions.

```
#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>
```

9.32.1 Detailed Description

File containing statistics template functions.

Definition in file [vec_stats.h](#).

Functions

- template<class vec_t >
double [vector_mean](#) (const size_t n, vec_t &data)
Compute the mean of the first n elements of a vector.
- template<class vec_t >
double [vector_variance_fmean](#) (const size_t n, vec_t &data, double mean)
Variance.
- template<class vec_t >
double [vector_stddev_fmean](#) (const size_t n, vec_t &data, double mean)
Standard deviation.
- template<class vec_t >
double [vector_variance](#) (const size_t n, vec_t &data, double mean)
Compute the variance of the first n elements of a vector given the mean mean.
- template<class vec_t >
double [vector_variance](#) (const size_t n, vec_t &data)
Variance.
- template<class vec_t >
double [vector_stddev](#) (const size_t n, vec_t &data)

Standard deviation.

- template<class vec_t >
double [vector_stddev](#) (const size_t n, vec_t &data, double mean)
Standard deviation.
- template<class vec_t >
double [vector_absdev](#) (const size_t n, vec_t &data, double mean)
Absolute deviation from the mean.
- template<class vec_t >
double [vector_absdev](#) (const size_t n, vec_t &data)
Absolute deviation from the mean.
- template<class vec_t >
double [vector_skew](#) (const size_t n, vec_t &data, double mean, double stddev)
Skewness.
- template<class vec_t >
double [vector_skew](#) (const size_t n, vec_t &data)
Skewness.
- template<class vec_t >
double [vector_kurtosis](#) (const size_t n, vec_t &data, double mean, double stddev)
Kurtosis.
- template<class vec_t >
double [vector_kurtosis](#) (const size_t n, vec_t &data)
Kurtosis.
- template<class vec_t >
double [vector_lag1_autocorr](#) (const size_t n, vec_t &data, double mean)
Lag1 autocorrelation.
- template<class vec_t >
double [vector_lag1_autocorr](#) (const size_t n, vec_t &data)
Lag1 autocorrelation.
- template<class vec_t >
double [vector_covariance](#) (const size_t n, vec_t &data1, vec_t &data2, double mean1, double mean2)
Covariance.
- template<class vec_t >
double [vector_covariance](#) (const size_t n, vec_t &data1, vec_t &data2)
Covariance.
- template<class vec_t >
double [vector_correlation](#) (const size_t n, vec_t &data1, vec_t &data2)
Pearson's correlation.
- template<class vec_t >
double [vector_pvariance](#) (const size_t n1, vec_t &data1, const size_t n2, vec_t &data2)
Pooled variance.
- template<class vec_t >
double [vector_quantile_sorted](#) (const size_t n, vec_t &data, const double f)
Quantile.
- template<class vec_t >
double [vector_median_sorted](#) (const size_t n, vec_t &data)
Quantile.

9.32.2 Function Documentation

9.32.2.1 double [vector_mean](#) (const size_t n, vec_t & data) [inline]

Compute the mean 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 51 of file vec_stats.h.

9.32.2.2 double vector_variance (const size_t n, vec_t & data, double mean) [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 87 of file [vec_stats.h](#).

9.33 vector.h File Reference

File for generic vector functions.

```
#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>
```

9.33.1 Detailed Description

File for generic vector functions.

For a more general discussion of vectors and matrices in *O₂scl*, see the [Arrays, Vectors, Matrices and Tensors](#) of the User's Guide.

For overloaded operators involving vectors and matrices, see [vec_arith.h](#). For statistics operations not included here, see [vec_stats.h](#) in the directory `src/other`. For functions and classes which are specific to C-style arrays, see [array.h](#). Also related are the matrix output functions, [matrix_out\(\)](#), [matrix_cx_out_paren\(\)](#), and [matrix_out_paren\(\)](#) which are defined in [columnify.h](#) because they utilize the class [columnify](#) to format the output.

For functions which search for a value in an ordered vector, see the class [search_vec](#).

Definition in file [vector.h](#).

Functions

- `template<class vec_t, class vec2_t >`
void [vector_copy](#) (size_t N, vec_t &src, vec2_t &dest)
Naive vector copy.
- `template<class mat_t, class mat2_t >`
void [matrix_copy](#) (size_t M, size_t N, mat_t &src, mat2_t &dest)
Naive matrix copy.
- `template<class vec_t, class vec2_t >`
void [vector_cx_copy_gsl](#) (size_t N, vec_t &src, vec2_t &dest)
GSL complex vector copy.
- `template<class mat_t, class mat2_t >`
void [matrix_cx_copy_gsl](#) (size_t M, size_t N, mat_t &src, mat2_t &dest)
GSL complex matrix copy.
- `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 vec_t >`
`double vector_max (const size_t n, vec_t &data)`
Compute the maximum of the first n elements of a vector.
- `template<class vec_t >`
`double vector_min (const size_t n, vec_t &data)`
Compute the minimum of the first n elements of a vector.
- `template<class vec_t >`
`int vector_minmax (const size_t n, vec_t &data, double &min, double &max)`
Compute the minimum and maximum of the first n elements of a vector.
- `template<class vec_t >`
`size_t vector_max_index (const size_t n, vec_t &data, double &max)`
Compute the maximum of the first n elements of a vector.
- `template<class vec_t >`
`int vector_min_index (const size_t n, vec_t &data, double &min)`
Compute the minimum of the first n elements of a vector.
- `template<class vec_t >`
`int vector_minmax_index (const size_t n, vec_t &data, double &min, size_t &ix, double &max, size_t &ix2)`
Compute the minimum and maximum of the first n elements of a vector.
- `template<class vec_t >`
`double vector_sum (const size_t n, vec_t &data)`
Compute the sum of the first n elements of a vector.
- `template<class data_t, class vec_t >`
`int vector_reverse (const size_t n, vec_t &data)`
Reverse a vector.

9.33.2 Function Documentation

9.33.2.1 `void matrix_copy (size_t M, size_t N, mat_t &src, mat2_t &dest)` [inline]

Naive matrix copy.

Note:

This ordering is reversed from the GSL function `gsl_matrix_memcpy`. This is to be used with

```
matrix_copy(N, source, destination);
```

instead of

```
gsl_matrix_memcpy(destination, source);
```

Definition at line 91 of file `vector.h`.

9.33.2.2 `void matrix_cx_copy_gsl (size_t M, size_t N, mat_t &src, mat2_t &dest)` [inline]

GSL complex matrix copy.

Idea for future

At present this works only with complex types based directly on the GSL complex format. This could be improved.

Definition at line 120 of file `vector.h`.

9.33.2.3 void vector_copy (size_t *N*, vec_t & *src*, vec2_t & *dest*) [inline]

Naive vector copy.

Note:

This ordering is reversed from the GSL function `gsl_vector_memcpy`. This is to be used with

```
vector_copy (N, source, destination);
```

instead of

```
gsl_vector_memcpy (destination, source);
```

Definition at line 73 of file vector.h.

9.33.2.4 void vector_cx_copy_gsl (size_t *N*, vec_t & *src*, vec2_t & *dest*) [inline]

GSL complex vector copy.

Idea for future

At present this works only with complex types based directly on the GSL complex format. This could be improved.

Definition at line 106 of file vector.h.

9.33.2.5 int vector_out (std::ostream & *os*, size_t *n*, vec_t & *v*, bool *endline* = false) [inline]

Output a vector to a stream.

No trailing space is output after the last element, and an `endline` is output only if `endline` is set to `true`. If the parameter `n` is zero, this function silently does nothing.

Note that the `O2scl` vector classes also have their own `operator<<()` defined for them.

Definition at line 140 of file vector.h.

9.33.2.6 int vector_rotate (const size_t *n*, vec_t & *data*, size_t *k*) [inline]

"Rotate" a vector so that the `k`th element is now the beginning

This is a generic template function which will work for any types `data_t` and `vec_t` for which

- `data_t` has an `operator=`
- `vec_t::operator[]` returns a reference to an object of type `data_t`

Definition at line 229 of file vector.h.

9.33.2.7 int vector_sort (const size_t *n*, vec_t & *data*) [inline]

Sort a vector.

This is a generic sorting template function. It will work for any types `data_t` and `vec_t` for which

- `data_t` has an `operator=`
- `data_t` has a less than operator to compare elements
- `vec_t::operator[]` returns a reference to an object of type `data_t`

In particular, it will work with [ovector](#), [uvector](#), [ovector_int](#), [uvector_int](#) (and other related O₂scl vector classes), the STL template class `std::vector`, and arrays and pointers of numeric, character, and string objects.

For example,

```
std::string list[3]={"dog", "cat", "fox"};
vector_sort<std::string, std::string[3]>(3,list);
```

Definition at line 193 of file `vector.h`.

9.33.2.8 `double vector_sum (const size_t n, vec_t & data)` [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 363 of file `vector.h`.

Index

accumulate_distribution
 gsl_vegas, 326
adapt_step, 112
 astep, 113
 astep_derivs, 113
 set_step, 113
add_col_from_table
 table, 534
add_type
 io_manager, 334
akima_interp, 114
 init, 115
akima_peri_interp, 115
align
 columnify, 165
allocate
 gsl_mmin_conf, 285
 o2scl_hybrid_state_t, 401
 ovector_tlate, 489
alpha
 gsl_miser, 276
 gsl_vegas, 327
anneal_mt, 116
 print_iter, 118
apply
 permutation, 493
apply_givens_lq
 o2scl_linalg, 108
apply_givens_qr
 o2scl_linalg, 108
apply_inverse
 permutation, 493
array.h, 593
 delete_2d_array, 595
 delete_array, 595
 new_2d_array, 595
 new_array, 596
array_2d_alloc, 118
array_2d_col, 119
array_2d_row, 119
array_alloc, 120
array_const_reverse, 120
array_const_subvector, 121
array_const_subvector_reverse, 122
array_interp, 122
array_interp_vec, 123
array_reverse, 123
array_subvector, 124
array_subvector_reverse, 125
astep
 adapt_step, 113
 gsl_astep, 234
 nonadapt_step, 399
astep_derivs
 adapt_step, 113
 gsl_astep, 234
 nonadapt_step, 400
avoid_nonzero
 gsl_mmin_simp, 292
 gsl_mmin_simp2, 295
base_interp, 125
 min_size, 127
base_interp_mgr, 127
base_ioc, 128
bin_size, 128
 calc_bin, 129
binary_in_file, 129
binary_out_file, 131
binary_to_hex
 misc.h, 614
bool_io_type, 132
bracket
 minimize, 358
bracket_step
 root_bkt, 511
bsearch_dec
 search_vec, 513
bsearch_inc
 search_vec, 513
btos
 string_conv.h, 627
calc
 deriv, 186
calc2_vector
 eqi_deriv, 191
calc3_vector
 eqi_deriv, 191
calc_bin
 bin_size, 129
calc_contours
 contour, 174
calc_err_int
 cern_deriv, 138
 deriv, 186
calc_Hv
 ool_constr_mmin, 449
calc_int
 deriv, 186
calc_vector
 eqi_deriv, 191
capacity
 ovector_const_view_tlate, 472
cblas_base.h, 596

- central_deriv
 - gsl_deriv, 242
- cern_adapt, 133
 - nseg, 134
- cern_cauchy, 134
- cern_cubic_real_coeff, 136
- cern_deriv, 137
 - calc_err_int, 138
- cern_gauss, 138
 - integ, 140
- cern_gauss56, 140
 - integ_err, 141
- cern_minimize, 141
 - min_bkt, 142
 - set_delta, 142
- cern_mroot, 143
 - eps, 145
 - get_info, 144
 - maxf, 145
- cern_mroot_root, 145
 - eps, 147
 - get_info, 147
 - maxf, 147
- cern_quartic_real_coeff, 148
- cern_root, 148
 - mode, 150
 - set_mode, 149
 - solve_bkt, 150
- change_box_coord
 - gsl_vegas, 326
- change_direction
 - gsl_mmin_wrapper, 298
- char_io_type, 150
- chisq
 - gsl_vegas, 327
- cinput, 151
- clear_data
 - table, 534
 - table3d, 542
- clear_types
 - io_type_info, 339
- cli, 152
 - cli_gets, 155
 - gnu_intro, 156
 - process_args, 155
 - separate, 155
 - set_alias, 155
 - set_comm_option, 155
 - set_verbose, 156
- cli_gets
 - cli, 155
- cli_readline, 156
- clock_seed
 - rnga, 509
- cmd_line_arg, 157
- collection, 157
 - disown, 161
 - get_void, 161
 - out_one, 161, 162
 - remove, 162
 - rewrite, 162
- collection.h, 597
- collection::iterator, 162
- collection::type_iterator, 163
- collection_entry, 164
- cols
 - omatrix_const_view_tlate, 437
 - omatrix_cx_view_tlate, 442
 - umatrix_const_view_tlate, 570
 - umatrix_cx_view_tlate, 575
- columnify, 164
 - align, 165
- columnify.h, 600
 - matrix_out, 600
 - matrix_out_paren, 600
- comm_option_funct, 165
- comm_option_mfptra, 166
- comm_option_s, 167
- comp_gen_inte, 168
- composite_inte, 169
- compute_size
 - gsl_mmin_simp2, 294
- constraint
 - minimize.h, 612
- cont_constraint
 - minimize.h, 612
- cont_lower_bound
 - minimize.h, 612
- contour, 171
 - calc_contours, 174
 - get_data, 174
 - regrid_data, 174
 - set_data, 174
 - set_levels, 174
- contour_line, 175
- contract_by_best
 - gsl_mmin_simp, 291
 - gsl_mmin_simp2, 294
- convert_units, 175
- convert_units::unit_t, 177
- count_words
 - misc.h, 614
- coutput, 177
 - npointers, 178
 - pointer_lookup, 178
- coutput::ltptra, 178
- cspline_interp, 179
 - init, 180
- cspline_peri_interp, 180
- cubic

- gsl_mmin_linmin, 288
- cubic_complex, 181
- cubic_real, 182
- cubic_real_coeff, 183
- cubic_std_complex, 183
- cx_arith.h, 601

- daxpy_hv_sub
 - o2scl_cblas, 99
- daxpy_subvec
 - o2scl_cblas, 99
- ddot_hv_sub
 - o2scl_cblas, 99
- ddot_subvec
 - o2scl_cblas, 99
- def_interp_mgr, 184
- delete_2d_array
 - array.h, 595
- delete_array
 - array.h, 595
- delete_column
 - table, 534
- deriv, 185
 - calc, 186
 - calc_err_int, 186
 - calc_int, 186
 - table, 534
- deriv2
 - table, 534
- deriv2_const
 - table, 534
- deriv::dpars, 187
- deriv_const
 - table, 535
- deriv_vector
 - eqi_deriv, 191
- deuf_kchoice
 - gsl_bsimp, 237
- disown
 - collection, 161
- dither
 - gsl_miser, 277
- dnrm2_subcol
 - o2scl_cblas, 99
- dnrm2_subvec
 - o2scl_cblas, 99
- double_io_type, 187
- double_to_ieee_string
 - string_conv.h, 627
- dscal_subcol
 - o2scl_cblas, 100
- dscal_subvec
 - o2scl_cblas, 100
- dtos
 - string_conv.h, 627

- e2_gaussian
 - o2scl_const, 101
- e2_hlorentz
 - o2scl_const, 101
- e2_mkxa
 - o2scl_const, 101
- edge_crossings, 188
- eigen_tdiag
 - lanczos, 348
- eigenvalues
 - lanczos, 348
- end_line
 - text_out_file, 561
- eps
 - cern_mroot, 145
 - cern_mroot_root, 147
- eqi_deriv, 189
 - calc2_vector, 191
 - calc3_vector, 191
 - calc_vector, 191
 - deriv_vector, 191
 - set_npoints, 191
 - set_npoints2, 191
- err_hnd.h
 - gsl_continue, 605
 - gsl_ebadfunc, 606
 - gsl_ebadlen, 606
 - gsl_ebadtol, 606
 - gsl_ecache, 606
 - gsl_ediverge, 606
 - gsl_edom, 606
 - gsl_efactor, 606
 - gsl_efailed, 606
 - gsl_efault, 606
 - gsl_efilenotfound, 606
 - gsl_eindex, 606
 - gsl_einval, 606
 - gsl_eloss, 606
 - gsl_emaxiter, 606
 - gsl_ememtype, 606
 - gsl_enomem, 606
 - gsl_enoprog, 606
 - gsl_enoprogj, 606
 - gsl_enotfound, 606
 - gsl_enotsqr, 606
 - gsl_eof, 606
 - gsl_eovrflw, 606
 - gsl_erange, 606
 - gsl_eround, 606
 - gsl_erunaway, 606
 - gsl_esanity, 606
 - gsl_esing, 606
 - gsl_etable, 606
 - gsl_etol, 606
 - gsl_etolf, 606

- gsl_etolg, 606
- gsl_etolx, 606
- gsl_eundrflw, 606
- gsl_eunimpl, 606
- gsl_eunsup, 606
- gsl_ezerodiv, 606
- gsl_failure, 605
- gsl_outsidecons, 606
- gsl_success, 605
- err_base, 192
 - gsl_hnd, 193
- err_class, 193
 - set_mode, 194
- err_hnd.h, 603
 - error_update, 607
 - O2SCL_ASSERT, 605
 - O2SCL_CONV_RET, 605
 - O2SCL_ERR2, 605
 - O2SCL_ERR2_RET, 605
- error_update
 - err_hnd.h, 607
- estimate_frac
 - gsl_miser, 277
- ex_wk
 - gsl_bsimp, 238
- exact_jacobian, 195
- exact_jacobian::ej_parms, 196
- exit_on_fail
 - ode_iv_solve, 418
- fermi_function
 - misc.h, 614
- feval
 - gsl_inte_qng, 266
- file_detect, 196
 - open, 197
- find_inc_subset
 - smart_interp_vec, 525
- find_interval_inc
 - search_vec, 514
- find_interval_uncons
 - search_vec, 514
- find_subset
 - smart_interp, 523
- fit
 - fit_base, 198
 - fit_fix_pars, 200
 - gsl_fit, 245
 - min_fit, 356
- fit_base, 198
 - fit, 198
 - print_iter, 198
- fit_fix_pars, 199
 - fit, 200
- fit_funct, 200
- fit_funct_cmfp_ptr, 201
- fit_funct_fptr, 202
- fit_funct_mfp_ptr, 203
- fit_vfunct, 204
- fit_vfunct_cmfp_ptr, 204
- fit_vfunct_fptr, 205
- fit_vfunct_mfp_ptr, 206
- fmin
 - ool_mmin_gencan, 457
 - ool_mmin_pgrad, 458
 - ool_mmin_spg, 460
- format_float, 207
 - html_mode, 210
 - latex_mode, 210
- free
 - omatrix_cx_tlate, 441
 - omatrix_tlate, 445
 - ovector_cx_tlate, 481
 - ovector_tlate, 489
 - permutation, 493
 - umatrix_cx_tlate, 573
 - umatrix_tlate, 577
 - uvector_cx_tlate, 587
 - uvector_tlate, 591
- funct, 210
- funct_cmfp_ptr, 211
- funct_cmfp_ptr_noerr, 213
- funct_cmfp_ptr_nopar, 214
- funct_fptr, 215
- funct_fptr_noerr, 215
- funct_fptr_nopar, 216
- funct_mfp_ptr, 217
- funct_mfp_ptr_noerr, 218
- funct_mfp_ptr_nopar, 219
- gaussian_2d, 220
- gen_inte, 221
 - get_error, 222
 - ginteg, 222
 - ginteg_err, 222
- gen_test_number, 222
- get_coefficient
 - gsl_chebapp, 239
- get_data
 - contour, 174
- get_error
 - gen_inte, 222
 - inte, 331
 - multi_inte, 387
- get_info
 - cern_mroot, 144
 - cern_mroot_root, 147
- get_void
 - collection, 161
- ginteg
 - gen_inte, 222
- ginteg_err

- gen_inte, 222
- givens.h, 607
- givens_base.h, 607
- gnu_intro
 - cli, 156
- grad_funct, 223
- grad_funct_cmfp_ptr, 224
- grad_funct_fptr, 225
- grad_funct_mfp_ptr, 226
- grad_vfunct, 226
- grad_vfunct_cmfp_ptr, 227
- grad_vfunct_fptr, 228
- grad_vfunct_mfp_ptr, 229
- gradient, 229
- gradient_array, 230
- gsl_continue
 - err_hnd.h, 605
- gsl_ebadfunc
 - err_hnd.h, 606
- gsl_ebadlen
 - err_hnd.h, 606
- gsl_ebadtol
 - err_hnd.h, 606
- gsl_ecache
 - err_hnd.h, 606
- gsl_ediverge
 - err_hnd.h, 606
- gsl_edom
 - err_hnd.h, 606
- gsl_efactor
 - err_hnd.h, 606
- gsl_efailed
 - err_hnd.h, 606
- gsl_efault
 - err_hnd.h, 606
- gsl_efilenotfound
 - err_hnd.h, 606
- gsl_eindex
 - err_hnd.h, 606
- gsl_einval
 - err_hnd.h, 606
- gsl_eloss
 - err_hnd.h, 606
- gsl_emaxiter
 - err_hnd.h, 606
- gsl_ememtype
 - err_hnd.h, 606
- gsl_enomem
 - err_hnd.h, 606
- gsl_enoprog
 - err_hnd.h, 606
- gsl_enoprogj
 - err_hnd.h, 606
- gsl_enotfound
 - err_hnd.h, 606
- gsl_enotsqr
 - err_hnd.h, 606
- gsl_eof
 - err_hnd.h, 606
- gsl_eovrflw
 - err_hnd.h, 606
- gsl_erange
 - err_hnd.h, 606
- gsl_eround
 - err_hnd.h, 606
- gsl_erunaway
 - err_hnd.h, 606
- gsl_esanity
 - err_hnd.h, 606
- gsl_esing
 - err_hnd.h, 606
- gsl_etable
 - err_hnd.h, 606
- gsl_etol
 - err_hnd.h, 606
- gsl_etolf
 - err_hnd.h, 606
- gsl_etolg
 - err_hnd.h, 606
- gsl_etolx
 - err_hnd.h, 606
- gsl_eundrflw
 - err_hnd.h, 606
- gsl_eunimpl
 - err_hnd.h, 606
- gsl_eunsup
 - err_hnd.h, 606
- gsl_ezerodiv
 - err_hnd.h, 606
- gsl_failure
 - err_hnd.h, 605
- gsl_outsidecons
 - err_hnd.h, 606
- gsl_success
 - err_hnd.h, 605
- gsl_alloc_arrays
 - misc.h, 614
- gsl_anneal, 231
- gsl_astep, 233
 - astep, 234
 - astep_derivs, 234
- gsl_bsimp, 235
 - deuf_kchoice, 237
 - ex_wk, 238
 - poly_extrap, 237
 - step, 237
 - step_local, 237
- gsl_cgs, 81
- gsl_cgsm, 85
- gsl_chebapp, 238

- get_coefficient, 239
- init, 239
- set_coefficient, 239
- gsl_cubic_real_coeff, 240
- gsl_poly_complex_solve_cubic2, 240
- gsl_deriv, 241
 - central_deriv, 242
 - h, 242
 - h_opt, 242
- gsl_fft, 242
- gsl_fit, 243
 - fit, 245
- gsl_fit::func_par, 245
- gsl_hnd
 - err_base, 193
- gsl_inte, 246
- gsl_inte_cheb, 246
 - gsl_integration_qcheb, 247
- gsl_inte_kronrod, 248
 - gsl_integration_qk_o2scl, 249
- gsl_inte_qag, 249
 - set_key, 251
- gsl_inte_qagi, 251
 - integ, 252
 - integ_err, 252
- gsl_inte_qagil, 252
 - integ, 253
 - integ_err, 253
- gsl_inte_qagiu, 254
 - integ, 255
 - integ_err, 255
- gsl_inte_qags, 255
- gsl_inte_qawc, 256
- gsl_inte_qawf_cos, 258
- gsl_inte_qawf_sin, 259
- gsl_inte_qawo_cos, 260
- gsl_inte_qawo_sin, 262
- gsl_inte_qaws, 263
- gsl_inte_qng, 265
 - feval, 266
 - integ_err, 266
- gsl_inte_singular, 266
 - qags, 267
 - qelg, 267
- gsl_inte_singular::extrapolation_table, 268
- gsl_inte_table, 269
 - qpsrt, 271
 - retrieve, 271
- gsl_inte_transform, 271
 - gsl_integration_qk_o2scl, 272
- gsl_integration_qcheb
 - gsl_inte_cheb, 247
- gsl_integration_qk_o2scl
 - gsl_inte_kronrod, 249
 - gsl_inte_transform, 272
- gsl_min_brent, 273
 - min_bkt, 274
- gsl_miser, 274
 - alpha, 276
 - dither, 277
 - estimate_frac, 277
 - min_calls, 277
 - min_calls_per_bisection, 277
- gsl_mks, 89
- gsl_mkssa, 93
- gsl_mmin_base, 278
 - intermediate_point, 280
 - minimize, 280
- gsl_mmin_bfgs2, 280
- gsl_mmin_bfgs2_array, 282
- gsl_mmin_conf, 283
 - allocate, 285
 - set, 285
- gsl_mmin_conf_array, 285
- gsl_mmin_conp, 286
- gsl_mmin_conp_array, 287
- gsl_mmin_linmin, 288
 - cubic, 288
 - interp_quad, 288
 - minimize, 288
- gsl_mmin_simp, 289
 - avoid_nonzero, 292
 - contract_by_best, 291
 - move_corner_err, 291
 - nmsimplex_size, 291
 - print_iter, 291
 - print_simplex, 292
- gsl_mmin_simp2, 292
 - avoid_nonzero, 295
 - compute_size, 294
 - contract_by_best, 294
 - print_iter, 295
 - print_simplex, 295
 - try_corner_move, 295
- gsl_mmin_wrap_base, 295
- gsl_mmin_wrapper, 296
 - change_direction, 298
- gsl_monte, 298
- gsl_mroot_hybrids, 299
 - iterate, 301
 - msolve_de, 301
 - shrink_step, 302
- gsl_num, 96
- gsl_ode_control, 302
- gsl_poly_complex_solve_cubic2
 - gsl_cubic_real_coeff, 240
- gsl_poly_real_coeff, 303
 - solve_rc, 304
- gsl_quadratic_real_coeff, 304
- gsl_quartic_real, 305

- gsl_quartic_real2, 306
- gsl_rk8pd, 307
 - step, 308
- gsl_rk8pd_fast, 308
 - step, 310
- gsl_rkck, 310
 - step, 311
- gsl_rkck_fast, 311
 - step, 313
- gsl_rkf45, 313
 - step, 314
- gsl_rkf45_fast, 314
 - step, 316
- gsl_rnga, 316
- gsl_root_brent, 317
 - iterate, 318
 - set, 318
 - solve_bkt, 318
- gsl_root_stef, 319
 - iterate, 320
 - set, 320
- gsl_series, 321
- gsl_vegas, 323
 - accumulate_distribution, 326
 - alpha, 327
 - change_box_coord, 326
 - chisq, 327
 - random_point, 326
 - vegas_minteg_err, 326
- h
 - gsl_deriv, 242
- h_opt
 - gsl_deriv, 242
- has_minus_sign
 - string_conv.h, 627
- hh_base.h, 608
- householder_base.h, 608
- householder_hm_sub
 - o2scl_linalg, 108
- householder_hv_sub
 - o2scl_linalg, 108
- householder_transform_subcol
 - o2scl_linalg, 108
- html_mode
 - format_float, 210
- hybrids_base, 327
- in_file_format, 328
- init
 - akima_interp, 115
 - cspline_interp, 180
 - gsl_chebapp, 239
- init_column
 - table, 535
- init_slice
 - table3d, 542
- inside
 - pinside, 494
- int_io_type, 329
- inte, 330
 - get_error, 331
 - last_conv, 331
- integ
 - cern_gauss, 140
 - gsl_inte_qagi, 252
 - gsl_inte_qagil, 253
 - gsl_inte_qagiu, 255
 - table, 535
- integ_const
 - table, 535
- integ_err
 - cern_gauss56, 141
 - gsl_inte_qagi, 252
 - gsl_inte_qagil, 253
 - gsl_inte_qagiu, 255
 - gsl_inte_qng, 266
- intermediate_point
 - gsl_mmin_base, 280
- interp
 - planar_intp, 497
 - smart_interp, 523
 - table, 535
- interp_const
 - table, 536
- interp_quad
 - gsl_mmin_linmin, 288
- interpolate
 - tensor_grid, 553
- intersect
 - pinside, 494
- intl_allocate
 - ovector_tlate, 489
- intl_init
 - ovector_tlate, 490
- io_base, 331
 - io_base, 333
 - io_base, 333
 - pointer_out, 333
- io_manager, 333
 - add_type, 334
- io_tlate, 335
 - stat_input, 338
- io_type_info, 338
 - clear_types, 339
 - remove_type, 339
- io_vtlate, 339
 - stat_input, 340
- is_column
 - table, 536
- is_owner

- omatrix_const_view_tlate, 437
- ovector_const_view_tlate, 472
- iterate
 - gsl_mroot_hybrids, 301
 - gsl_root_brent, 318
 - gsl_root_stef, 320
- jac_funct, 341
- jac_funct_cmfp_ptr, 341
- jac_funct_fptr, 342
- jac_funct_mfp_ptr, 343
- jac_vfunct, 344
- jac_vfunct_cmfp_ptr, 344
- jac_vfunct_fptr, 345
- jac_vfunct_mfp_ptr, 346
- jacobian, 346
- lanczos, 347
 - eigen_tdiag, 348
 - eigenvalues, 348
 - product, 349
- last_conv
 - inte, 331
 - minimize, 359
 - mroot, 377
 - multi_min, 388
 - root, 510
- latex_mode
 - format_float, 210
- lib_settings
 - lib_settings.h, 610
- lib_settings.h, 609
 - lib_settings, 610
- lib_settings_class, 349
- linear_interp, 350
- long_io_type, 353
- lookup
 - ovector_const_view_tlate, 472
 - uvector_const_view_tlate, 583
- lookup_column
 - table, 536
- lower_bound
 - minimize.h, 612
- lu_base.h, 610
- LU_decomp
 - o2scl_linalg, 109
- LU_det
 - o2scl_linalg, 109
- LU_invert
 - o2scl_linalg, 109
- LU_lndet
 - o2scl_linalg, 109
- LU_refine
 - o2scl_linalg, 109
- LU_sgndet
 - o2scl_linalg, 110
- LU_solve
 - o2scl_linalg, 110
- LU_svx
 - o2scl_linalg, 110
- mass_alpha
 - o2scl_fm, 103
- matrix_copy
 - vector.h, 652
- matrix_cx_copy_gsl
 - vector.h, 652
- matrix_out
 - columnify.h, 600
- matrix_out_paren
 - columnify.h, 600
- matrix_slice
 - tensor, 547
- max
 - ovector_const_view_tlate, 472
- max_index
 - ovector_const_view_tlate, 473
- maxf
 - cern_mroot, 145
 - cern_mroot_root, 147
- mcarlo_inte, 354
- min
 - minimize, 358
 - minimize_bkt, 360
 - minimize_de, 361
 - ovector_const_view_tlate, 473
- min_bkt
 - cern_minimize, 142
 - gsl_min_brent, 274
 - minimize, 358
 - minimize_bkt, 360
 - minimize_de, 361
- min_calls
 - gsl_miser, 277
- min_calls_per_bisection
 - gsl_miser, 277
- min_de
 - minimize, 358
 - minimize_bkt, 360
 - minimize_de, 361
- min_fit, 354
 - fit, 356
- min_fit::func_par, 356
- min_index
 - ovector_const_view_tlate, 473
- min_size
 - base_interp, 127
- minimize, 357
 - bracket, 358
 - gsl_mmin_base, 280
 - gsl_mmin_linmin, 288
 - last_conv, 359

- min, 358
- min_bkt, 358
- min_de, 358
- print_iter, 359
- minimize.h, 611
 - constraint, 612
 - cont_constraint, 612
 - cont_lower_bound, 612
 - lower_bound, 612
- minimize_bkt, 359
 - min, 360
 - min_bkt, 360
 - min_de, 360
- minimize_de, 360
 - min, 361
 - min_bkt, 361
 - min_de, 361
- misc.h, 613
 - binary_to_hex, 614
 - count_words, 614
 - fermi_function, 614
 - gsl_alloc_arrays, 614
 - screenify, 614
- mm_funct, 362
- mm_funct_cmfp_ptr, 362
- mm_funct_cmfp_ptr_nopar, 363
- mm_funct_fp_ptr, 364
- mm_funct_fp_ptr_nopar, 365
- mm_funct_gsl, 366
- mm_funct_mfp_ptr, 367
- mm_funct_mfp_ptr_nopar, 368
- mm_vfunct, 369
- mm_vfunct_cmfp_ptr, 369
- mm_vfunct_cmfp_ptr_nopar, 370
- mm_vfunct_fp_ptr, 371
- mm_vfunct_fp_ptr_nopar, 372
- mm_vfunct_gsl, 373
- mm_vfunct_mfp_ptr, 373
- mm_vfunct_mfp_ptr_nopar, 374
- mmin
 - ool_constr_mmin, 449
- mmin_fix
 - multi_min_fix, 390
- mode
 - cern_root, 150
- move_corner_err
 - gsl_mmin_simp, 291
- mroot, 375
 - last_conv, 377
 - msolve_de, 377
 - print_iter, 377
- msolve_de
 - gsl_mroot_hybrids, 301
 - mroot, 377
- multi_funct, 377
 - operator(), 378
- multi_funct_cmfp_ptr, 378
 - operator(), 379
- multi_funct_cmfp_ptr_noerr, 379
 - operator(), 380
- multi_funct_fp_ptr, 380
 - operator(), 381
- multi_funct_fp_ptr_noerr, 381
 - operator(), 382
- multi_funct_gsl, 382
 - operator(), 383
- multi_funct_mfp_ptr, 383
 - operator(), 384
- multi_funct_mfp_ptr_noerr, 384
 - operator(), 385
- multi_inte, 386
 - get_error, 387
- multi_min, 387
 - last_conv, 388
 - print_iter, 388
- multi_min_fix, 389
 - mmin_fix, 390
- multi_vfunct, 390
 - operator(), 391
- multi_vfunct_cmfp_ptr, 391
 - operator(), 392
- multi_vfunct_cmfp_ptr_noerr, 392
 - operator(), 393
- multi_vfunct_fp_ptr, 393
 - operator(), 394
- multi_vfunct_fp_ptr_noerr, 394
 - operator(), 395
- multi_vfunct_gsl, 395
 - operator(), 396
- multi_vfunct_mfp_ptr, 396
 - operator(), 397
- multi_vfunct_mfp_ptr_noerr, 397
 - operator(), 398
- new_2d_array
 - array.h, 595
- new_array
 - array.h, 596
- new_row
 - table, 536
- nmsimplex_size
 - gsl_mmin_simp, 291
- nonadapt_step, 399
 - astep, 399
 - astep_derivs, 400
- norm
 - ovector_const_view_tlate, 473
- npoiners
 - coutput, 178
- nseg
 - cern_adapt, 134

- o2scl, 97
- O2SCL_ASSERT
 - err_hnd.h, 605
- o2scl_cblas, 97
 - daxpy_hv_sub, 99
 - daxpy_subvec, 99
 - ddot_hv_sub, 99
 - ddot_subvec, 99
 - dnrm2_subcol, 99
 - dnrm2_subvec, 99
 - dscal_subcol, 100
 - dscal_subvec, 100
- o2scl_cblas_paren, 100
- o2scl_const, 100
 - e2_gaussian, 101
 - e2_hlorentz, 101
 - e2_mkasa, 101
- O2SCL_CONV_RET
 - err_hnd.h, 605
- O2SCL_ERR2
 - err_hnd.h, 605
- O2SCL_ERR2_RET
 - err_hnd.h, 605
- o2scl_fm, 102
 - mass_alpha, 103
- o2scl_hybrid_state_t, 400
 - allocate, 401
- o2scl_input
 - user_io.h, 634, 635
- o2scl_input_name
 - user_io.h, 635
- o2scl_input_name_text
 - user_io.h, 635
- o2scl_input_text
 - user_io.h, 635, 636
- o2scl_inte_qag_coeffs, 103
- o2scl_inte_qng_coeffs, 104
 - w10, 105
 - w21a, 105
 - w21b, 105
 - w43a, 105
 - w43b, 105
 - w87a, 105
 - w87b, 105
 - x1, 105
 - x2, 105
 - x3, 105
 - x4, 106
- o2scl_interp, 402
- o2scl_interp_vec, 403
- o2scl_linalg, 106
 - apply_givens_lq, 108
 - apply_givens_qr, 108
 - householder_hm_sub, 108
 - householder_hv_sub, 108
 - householder_transform_subcol, 108
 - LU_decomp, 109
 - LU_det, 109
 - LU_invert, 109
 - LU_ldet, 109
 - LU_refine, 109
 - LU_sgndet, 110
 - LU_solve, 110
 - LU_svx, 110
 - QR_update, 110
 - solve_cyc_tridiag_nonsym, 110
 - solve_cyc_tridiag_sym, 111
 - solve_tridiag_nonsym, 111
 - solve_tridiag_sym, 111
- o2scl_linalg::gsl_solver_HH, 321
- o2scl_linalg::gsl_solver_LU, 322
- o2scl_linalg::gsl_solver_QR, 322
- o2scl_linalg::linear_solver, 350
- o2scl_linalg::linear_solver_hh, 351
- o2scl_linalg::linear_solver_lu, 352
- o2scl_linalg::linear_solver_qr, 352
- o2scl_linalg_paren, 112
- O2SCL_OP_CMAT_CVEC_MULT
 - vec_arith.h, 643
- O2SCL_OP_CX_DOT_PROD
 - vec_arith.h, 643
- O2SCL_OP_DOT_PROD
 - vec_arith.h, 644
- O2SCL_OP_MAT_VEC_MULT
 - vec_arith.h, 644
- O2SCL_OP_SCA_VEC_MULT
 - vec_arith.h, 644
- O2SCL_OP_TRANS_MULT
 - vec_arith.h, 644
- O2SCL_OP_VEC_MAT_MULT
 - vec_arith.h, 645
- O2SCL_OP_VEC_SCA_MULT
 - vec_arith.h, 645
- O2SCL_OP_VEC_VEC_ADD
 - vec_arith.h, 645
- O2SCL_OP_VEC_VEC_EQUAL
 - vec_arith.h, 646
- O2SCL_OP_VEC_VEC_NEQUAL
 - vec_arith.h, 646
- O2SCL_OP_VEC_VEC_PRO
 - vec_arith.h, 646
- O2SCL_OP_VEC_VEC_SUB
 - vec_arith.h, 647
- O2SCL_OPSRC_CMAT_CVEC_MULT
 - vec_arith.h, 647
- O2SCL_OPSRC_CX_DOT_PROD
 - vec_arith.h, 647
- O2SCL_OPSRC_DOT_PROD
 - vec_arith.h, 647
- O2SCL_OPSRC_MAT_VEC_MULT

- vec_arith.h, 647
- O2SCL_OPSRC_SCA_VEC_MULT
 - vec_arith.h, 647
- O2SCL_OPSRC_TRANS_MULT
 - vec_arith.h, 648
- O2SCL_OPSRC_VEC_MAT_MULT
 - vec_arith.h, 648
- O2SCL_OPSRC_VEC_SCA_MULT
 - vec_arith.h, 648
- O2SCL_OPSRC_VEC_VEC_ADD
 - vec_arith.h, 648
- O2SCL_OPSRC_VEC_VEC_EQUAL
 - vec_arith.h, 648
- O2SCL_OPSRC_VEC_VEC_NEQUAL
 - vec_arith.h, 648
- O2SCL_OPSRC_VEC_VEC_PRO
 - vec_arith.h, 648
- O2SCL_OPSRC_VEC_VEC_SUB
 - vec_arith.h, 649
- o2scl_output
 - user_io.h, 636
- o2scl_output_text
 - user_io.h, 637
- ode_bv_multishoot, 405
- ode_bv_shoot, 406
- ode_bv_solve, 408
- ode_funct, 409
- ode_funct_cmfp_ptr, 409
- ode_funct_fp_ptr, 410
- ode_funct_mfp_ptr, 411
- ode_it_funct, 412
- ode_it_funct_fp_ptr, 412
- ode_it_funct_mfp_ptr, 413
- ode_it_solve, 414
- ode_iv_solve, 415
 - exit_on_fail, 418
 - solve_final_value, 417
 - solve_final_value_derivs, 417
 - solve_grid, 417
 - solve_grid_derivs, 417
 - solve_table, 418
- ode_jac_funct, 418
- ode_jac_funct_cmfp_ptr, 419
- ode_jac_funct_fp_ptr, 420
- ode_jac_funct_mfp_ptr, 421
- ode_jac_vfunct, 422
- ode_jac_vfunct_cmfp_ptr, 423
- ode_jac_vfunct_fp_ptr, 423
- ode_jac_vfunct_mfp_ptr, 424
- ode_vfunct, 425
- ode_vfunct_cmfp_ptr, 426
- ode_vfunct_fp_ptr, 427
- ode_vfunct_mfp_ptr, 428
- odestep, 428
 - step, 429
- ofvector, 429
- ofvector_cx, 430
- omatrix_alloc, 431
- omatrix_array_tlate, 431
- omatrix_base_tlate, 432
- omatrix_col_tlate, 433
- omatrix_const_col_tlate, 434
- omatrix_const_diag_tlate, 435
- omatrix_const_row_tlate, 435
- omatrix_const_view_tlate, 436
 - cols, 437
 - is_owner, 437
 - rows, 437
 - tda, 437
- omatrix_cx_col_tlate, 437
- omatrix_cx_const_col_tlate, 438
- omatrix_cx_const_row_tlate, 439
- omatrix_cx_row_tlate, 439
- omatrix_cx_tlate, 440
 - free, 441
- omatrix_cx_tlate.h, 615
- omatrix_cx_view_tlate, 441
 - cols, 442
 - rows, 442
 - tda, 442
- omatrix_diag_tlate, 443
- omatrix_row_tlate, 443
- omatrix_tlate, 444
 - free, 445
- omatrix_tlate.h, 616
 - operator<<, 618
- omatrix_view_tlate, 446
- ool_constr_mmin, 447
 - calc_Hv, 449
 - mmin, 449
- ool_hfunct, 450
- ool_hfunct_fp_ptr, 450
- ool_hfunct_mfp_ptr, 451
- ool_hvfunct, 452
- ool_hvfunct_fp_ptr, 452
- ool_hvfunct_mfp_ptr, 453
- ool_mmin_gencan, 454
 - fmin, 457
- ool_mmin_pgrad, 457
 - fmin, 458
- ool_mmin_spg, 459
 - fmin, 460
- open
 - file_detect, 197
- operator<<
 - omatrix_tlate.h, 618
 - ovector_cx_tlate.h, 620
 - ovector_tlate.h, 623
 - permutation.h, 624
 - umatrix_cx_tlate.h, 631

- umatrix_tlate.h, 633
- uvector_cx_tlate.h, 638
- uvector_tlate.h, 640
- operator*=
 - ovector_base_tlate, 466
- operator()
 - multi_funct, 378
 - multi_funct_cmfp_ptr, 379
 - multi_funct_cmfp_ptr_noerr, 380
 - multi_funct_fptr, 381
 - multi_funct_fptr_noerr, 382
 - multi_funct_gsl, 383
 - multi_funct_mfp_ptr, 384
 - multi_funct_mfp_ptr_noerr, 385
 - multi_vfunct, 391
 - multi_vfunct_cmfp_ptr, 392
 - multi_vfunct_cmfp_ptr_noerr, 393
 - multi_vfunct_fptr, 394
 - multi_vfunct_fptr_noerr, 395
 - multi_vfunct_gsl, 396
 - multi_vfunct_mfp_ptr, 397
 - multi_vfunct_mfp_ptr_noerr, 398
- operator+=
 - ovector_base_tlate, 466
- operator-=
 - ovector_base_tlate, 466
- ordered_lookup
 - search_vec, 514
 - table, 537
- other_todos_and_bugs, 460
- out_file_format, 462
- out_one
 - collection, 161, 162
- ovector_alloc, 463
- ovector_array_stride_tlate, 463
- ovector_array_tlate, 464
- ovector_base_tlate, 464
 - operator*=, 466
 - operator+=, 466
 - operator-=, 466
 - set_all, 466
- ovector_const_array_stride_tlate, 467
- ovector_const_array_tlate, 467
- ovector_const_reverse_tlate, 468
- ovector_const_subvector_reverse_tlate, 469
- ovector_const_subvector_tlate, 470
- ovector_const_view_tlate, 470
 - capacity, 472
 - is_owner, 472
 - lookup, 472
 - max, 472
 - max_index, 473
 - min, 473
 - min_index, 473
 - norm, 473
 - size, 473
 - stride, 473
- ovector_cx_array_stride_tlate, 474
- ovector_cx_array_tlate, 474
- ovector_cx_const_array_stride_tlate, 475
- ovector_cx_const_array_tlate, 476
- ovector_cx_const_subvector_tlate, 477
- ovector_cx_imag_tlate, 478
- ovector_cx_real_tlate, 479
- ovector_cx_subvector_tlate, 479
- ovector_cx_tlate, 480
 - free, 481
- ovector_cx_tlate.h, 618
 - operator<<, 620
- ovector_cx_view_tlate, 481
 - size, 484
 - stride, 484
- ovector_int_alloc, 484
- ovector_rev_tlate.h, 620
- ovector_reverse_tlate, 484
- ovector_subvector_reverse_tlate, 486
- ovector_subvector_tlate, 487
- ovector_tlate, 487
 - allocate, 489
 - free, 489
 - intl_allocate, 489
 - intl_init, 490
 - reserve, 490
- ovector_tlate.h, 621
 - operator<<, 623
- ovector_view_tlate, 490
 - set_all, 491
- permutation, 492
 - apply, 493
 - apply_inverse, 493
 - free, 493
 - size, 493
- permutation.h, 623
 - operator<<, 624
- pinside, 493
 - inside, 494
 - intersect, 494
- pinside::line, 495
- pinside::point, 495
- planar_intp, 495
 - interp, 497
- pointer_2d_alloc, 498
- pointer_alloc, 498
- pointer_input, 499
- pointer_lookup
 - coutput, 178
- pointer_out
 - io_base, 333
- pointer_output, 499
- poly.h, 624

- poly_complex, 500
 - solve_c, 500
 - poly_extrap
 - gsl_bsimp, 237
 - poly_real_coeff, 500
 - solve_rc, 501
 - polylog, 501
 - print_iter
 - anneal_mt, 118
 - fit_base, 198
 - gsl_mmin_simp, 291
 - gsl_mmin_simp2, 295
 - minimize, 359
 - mroot, 377
 - multi_min, 388
 - root, 510
 - sim_anneal, 516
 - print_simplex
 - gsl_mmin_simp, 292
 - gsl_mmin_simp2, 295
 - process_args
 - cli, 155
 - product
 - lanczos, 349
 - ptos
 - string_conv.h, 628
 - qags
 - gsl_inte_singular, 267
 - qelg
 - gsl_inte_singular, 267
 - qpstrt
 - gsl_inte_table, 271
 - qr_base.h, 626
 - QR_update
 - o2scl_linalg, 110
 - quadratic_complex, 503
 - quadratic_real, 503
 - quadratic_real_coeff, 504
 - quadratic_std_complex, 505
 - quartic_complex, 505
 - quartic_real, 506
 - quartic_real_coeff, 507
 - random_point
 - gsl_vegas, 326
 - regrid_data
 - contour, 174
 - remove
 - collection, 162
 - remove_type
 - io_type_info, 339
 - rename_slice
 - table3d, 543
 - report
 - test_mgr, 557
 - reserve
 - ovector_tlate, 490
 - reset_interp
 - twod_intp, 564
 - reset_list
 - table, 537
 - retrieve
 - gsl_inte_table, 271
 - rewrite
 - collection, 162
 - rnga, 508
 - clock_seed, 509
 - root, 509
 - last_conv, 510
 - print_iter, 510
 - root_bkt, 510
 - bracket_step, 511
 - root_de, 511
 - rows
 - omatrix_const_view_tlate, 437
 - omatrix_cx_view_tlate, 442
 - umatrix_const_view_tlate, 570
 - umatrix_cx_view_tlate, 575
 - screenify
 - misc.h, 614
 - search_vec, 512
 - bsearch_dec, 513
 - bsearch_inc, 513
 - find_interval_inc, 514
 - find_interval_uncons, 514
 - ordered_lookup, 514
 - separate
 - cli, 155
 - set
 - gsl_mmin_conf, 285
 - gsl_root_brent, 318
 - gsl_root_stef, 320
 - table, 537
 - set_alias
 - cli, 155
 - set_all
 - ovector_base_tlate, 466
 - ovector_view_tlate, 491
 - set_coefficient
 - gsl_chebapp, 239
 - set_comm_option
 - cli, 155
 - set_data
 - contour, 174
 - twod_intp, 564
 - set_delta
 - cern_minimize, 142
 - set_grid
 - tensor_grid, 553
 - set_interp
-

- twod_intp, 564
- set_key
 - gsl_inte_qag, 251
- set_levels
 - contour, 174
- set_mode
 - cern_root, 149
 - err_class, 194
- set_nlines
 - table, 537
- set_nlines_auto
 - table, 537
- set_npoints
 - eqi_deriv, 191
- set_npoints2
 - eqi_deriv, 191
- set_output_level
 - test_mgr, 557
- set_size
 - table3d, 543
- set_step
 - adapt_step, 113
- set_type
 - twod_eqi_intp, 562
- set_verbose
 - cli, 156
- set_xy
 - table3d, 543
- shrink_step
 - gsl_mroot_hybrids, 302
- sim_anneal, 515
 - print_iter, 516
- simple_grad, 516
- simple_grad_array, 517
- simple_jacobian, 517
- simple_quartic_complex, 519
- simple_quartic_real, 519
- size
 - ovector_const_view_tlate, 473
 - ovector_cx_view_tlate, 484
 - permutation, 493
 - uvector_const_view_tlate, 583
 - uvector_cx_view_tlate, 589
- size_of_exponent
 - string_conv.h, 628
- sma_interp, 520
- sma_interp_vec, 521
- smart_interp, 521
 - find_subset, 523
 - interp, 523
- smart_interp_vec, 523
 - find_inc_subset, 525
- solve_bkt
 - cern_root, 150
 - gsl_root_brent, 318
- solve_c
 - poly_complex, 500
- solve_cyc_tridiag_nonsym
 - o2scl_linalg, 110
- solve_cyc_tridiag_sym
 - o2scl_linalg, 111
- solve_final_value
 - ode_iv_solve, 417
- solve_final_value_derivs
 - ode_iv_solve, 417
- solve_grid
 - ode_iv_solve, 417
- solve_grid_derivs
 - ode_iv_solve, 417
- solve_rc
 - gsl_poly_real_coeff, 304
 - poly_real_coeff, 501
- solve_table
 - ode_iv_solve, 418
- solve_tridiag_nonsym
 - o2scl_linalg, 111
- solve_tridiag_sym
 - o2scl_linalg, 111
- stat_input
 - io_tlate, 338
 - io_vtlate, 340
- step
 - gsl_bsimp, 237
 - gsl_rk8pd, 308
 - gsl_rk8pd_fast, 310
 - gsl_rkck, 311
 - gsl_rkck_fast, 313
 - gsl_rkf45, 314
 - gsl_rkf45_fast, 316
 - odestep, 429
- step_local
 - gsl_bsimp, 237
- stob
 - string_conv.h, 628
- stod
 - string_conv.h, 628
- stoi
 - string_conv.h, 628
- stride
 - ovector_const_view_tlate, 473
 - ovector_cx_view_tlate, 484
- string_comp, 525
- string_conv.h, 626
 - btos, 627
 - double_to_ieee_string, 627
 - dtos, 627
 - has_minus_sign, 627
 - ptos, 628
 - size_of_exponent, 628
 - stob, 628

- stod, 628
- stoi, 628
- string_io_type, 526
- subtable
 - table, 537
- summary
 - table, 537
 - table3d, 543
- table, 526
 - add_col_from_table, 534
 - clear_data, 534
 - delete_column, 534
 - deriv, 534
 - deriv2, 534
 - deriv2_const, 534
 - deriv_const, 535
 - init_column, 535
 - integ, 535
 - integ_const, 535
 - interp, 535
 - interp_const, 536
 - is_column, 536
 - lookup_column, 536
 - new_row, 536
 - ordered_lookup, 537
 - reset_list, 537
 - set, 537
 - set_nlines, 537
 - set_nlines_auto, 537
 - subtable, 537
 - summary, 537
- table3d, 538
 - clear_data, 542
 - init_slice, 542
 - rename_slice, 543
 - set_size, 543
 - set_xy, 543
 - summary, 543
- table::col_s, 543
- table::sortd_s, 544
- table_units, 544
- tda
 - omatrix_const_view_tlate, 437
 - omatrix_cx_view_tlate, 442
- tensor, 546
 - matrix_slice, 547
 - tensor, 547
 - tensor_allocate, 547
 - vector_slice, 548
- tensor.h, 628
- tensor1, 548
- tensor2, 549
- tensor3, 550
- tensor4, 550
- tensor_allocate
 - tensor, 547
 - tensor_grid, 553
- tensor_grid, 551
 - interpolate, 553
 - set_grid, 553
 - tensor_allocate, 553
 - tensor_grid, 552
 - tensor_grid, 552
- tensor_grid2, 553
- tensor_grid3, 554
- tensor_grid4, 555
- test_mgr, 556
 - report, 557
 - set_output_level, 557
- text_in_file, 558
- text_out_file, 559
 - end_line, 561
 - text_out_file, 561
 - text_out_file, 561
- tridiag_base.h, 629
- try_corner_move
 - gsl_mmin_simp2, 295
- twod_eqi_intp, 562
 - set_type, 562
- twod_intp, 563
 - reset_interp, 564
 - set_data, 564
 - set_interp, 564
- ufmatrix, 565
- ufmatrix_cx, 565
- ufvector, 566
- umatrix_alloc, 566
- umatrix_base_tlate, 567
- umatrix_const_row_tlate, 568
- umatrix_const_view_tlate, 569
 - cols, 570
 - rows, 570
- umatrix_cx_alloc, 570
- umatrix_cx_const_row_tlate, 571
- umatrix_cx_row_tlate, 571
- umatrix_cx_tlate, 572
 - free, 573
- umatrix_cx_tlate.h, 630
 - operator<<, 631
- umatrix_cx_view_tlate, 573
 - cols, 575
 - rows, 575
- umatrix_row_tlate, 575
- umatrix_tlate, 576
 - free, 577
- umatrix_tlate.h, 631
 - operator<<, 633
- umatrix_view_tlate, 577
- user_io.h, 633
 - o2scl_input, 634, 635

- o2scl_input_name, 635
- o2scl_input_name_text, 635
- o2scl_input_text, 635, 636
- o2scl_output, 636
- o2scl_output_text, 637
- uvector_alloc, 578
- uvector_array_tlate, 579
- uvector_base_tlate, 579
- uvector_const_array_tlate, 581
- uvector_const_subvector_tlate, 581
- uvector_const_view_tlate, 582
 - lookup, 583
 - size, 583
- uvector_cx_array_tlate, 584
- uvector_cx_const_array_tlate, 584
- uvector_cx_const_subvector_tlate, 585
- uvector_cx_subvector_tlate, 586
- uvector_cx_tlate, 586
 - free, 587
- uvector_cx_tlate.h, 637
 - operator<=, 638
- uvector_cx_view_tlate, 587
 - size, 589
- uvector_int_alloc, 589
- uvector_subvector_tlate, 589
- uvector_tlate, 590
 - free, 591
- uvector_tlate.h, 638
 - operator<=, 640
- uvector_view_tlate, 591
- vec_arith.h, 640
 - O2SCL_OP_CMAT_CVEC_MULT, 643
 - O2SCL_OP_CX_DOT_PROD, 643
 - O2SCL_OP_DOT_PROD, 644
 - O2SCL_OP_MAT_VEC_MULT, 644
 - O2SCL_OP_SCA_VEC_MULT, 644
 - O2SCL_OP_TRANS_MULT, 644
 - O2SCL_OP_VEC_MAT_MULT, 645
 - O2SCL_OP_VEC_SCA_MULT, 645
 - O2SCL_OP_VEC_VEC_ADD, 645
 - O2SCL_OP_VEC_VEC_EQUAL, 646
 - O2SCL_OP_VEC_VEC_NEQUAL, 646
 - O2SCL_OP_VEC_VEC_PRO, 646
 - O2SCL_OP_VEC_VEC_SUB, 647
 - O2SCL_OP_SRC_CMAT_CVEC_MULT, 647
 - O2SCL_OP_SRC_CX_DOT_PROD, 647
 - O2SCL_OP_SRC_DOT_PROD, 647
 - O2SCL_OP_SRC_MAT_VEC_MULT, 647
 - O2SCL_OP_SRC_SCA_VEC_MULT, 647
 - O2SCL_OP_SRC_TRANS_MULT, 648
 - O2SCL_OP_SRC_VEC_MAT_MULT, 648
 - O2SCL_OP_SRC_VEC_SCA_MULT, 648
 - O2SCL_OP_SRC_VEC_VEC_ADD, 648
 - O2SCL_OP_SRC_VEC_VEC_EQUAL, 648
 - O2SCL_OP_SRC_VEC_VEC_NEQUAL, 648
- O2SCL_OP_SRC_VEC_VEC_PRO, 648
- O2SCL_OP_SRC_VEC_VEC_SUB, 649
- vec_stats.h, 649
 - vector_mean, 650
 - vector_variance, 650
- vector.h, 651
 - matrix_copy, 652
 - matrix_cx_copy_gsl, 652
 - vector_copy, 652
 - vector_cx_copy_gsl, 653
 - vector_out, 653
 - vector_rotate, 653
 - vector_sort, 653
 - vector_sum, 654
- vector_copy
 - vector.h, 652
- vector_cx_copy_gsl
 - vector.h, 653
- vector_mean
 - vec_stats.h, 650
- vector_out
 - vector.h, 653
- vector_rotate
 - vector.h, 653
- vector_slice
 - tensor, 548
- vector_sort
 - vector.h, 653
- vector_sum
 - vector.h, 654
- vector_variance
 - vec_stats.h, 650
- vegas_minteg_err
 - gsl_vegas, 326
- w10
 - o2scl_inte_qng_coeffs, 105
- w21a
 - o2scl_inte_qng_coeffs, 105
- w21b
 - o2scl_inte_qng_coeffs, 105
- w43a
 - o2scl_inte_qng_coeffs, 105
- w43b
 - o2scl_inte_qng_coeffs, 105
- w87a
 - o2scl_inte_qng_coeffs, 105
- w87b
 - o2scl_inte_qng_coeffs, 105
- word_io_type, 593
- x1
 - o2scl_inte_qng_coeffs, 105
- x2
 - o2scl_inte_qng_coeffs, 105
- x3

o2scl_inte_qng_coeffs, [105](#)
x4
o2scl_inte_qng_coeffs, [106](#)
