

fortran_mathematics_library
1.0.0

Generated by Doxygen 1.7.1

Tue Apr 26 2011 12:43:25

Contents

1 Fortran Mathematics Library	1
1.1 Introduction	1
2 Files Tree	3
3 Notes on building	5
3.1 Description	5
4 Run methods	7
5 Notes on make help	9
5.1 Description	9
6 Todo List	11
7 Directory Hierarchy	13
7.1 Directories	13
8 Modules Index	15
8.1 Modules List	15
9 Data Type Index	17
9.1 Data Types List	17
10 File Index	19
10.1 File List	19
11 Directory Documentation	21
11.1 /home/akc/Desktop/projet_fortran/fml/src/ Directory Reference	21
12 Module Documentation	23
12.1 mod_exception Module Reference	23
12.1.1 Detailed Description	23

12.1.2	Description	24
12.1.3	Function/Subroutine Documentation	24
12.1.3.1	e_error	24
12.1.3.2	e_print_err	25
12.2	mod_linear_equation Module Reference	25
12.2.1	Detailed Description	26
12.2.2	Description	27
12.2.3	Function/Subroutine Documentation	27
12.2.3.1	gsb_getBlock_m	27
12.2.3.2	gsb_index_blocks	28
12.2.3.3	leq_cg_elt	28
12.2.3.4	leq_Cholesky	29
12.2.3.5	leq_gaussJourdan	30
12.2.3.6	leq_gaussSeidel_block	30
12.2.3.7	leq_gaussSeidel_elt	31
12.2.3.8	leq_getBlock_v	32
12.2.3.9	leq_gradient_elt	32
12.2.3.10	leq_jacobi_elt	33
12.2.3.11	leq_leastSquare	34
12.2.3.12	leq_leastSquare_QR	35
12.2.3.13	leq_lu	35
12.2.3.14	leq_precond_diag	36
12.2.3.15	leq_precond_sor	36
12.2.3.16	leq_pseudoinverse	37
12.2.3.17	leq_qr	38
12.2.3.18	leq_solve	39
12.2.3.19	leq_solve_iter	39
12.2.3.20	leq_sor_elt	40
12.2.3.21	leq_thomas	41
12.2.3.22	leq_tril	42
12.2.3.23	leq_triu	42
12.2.3.24	m_destruct_soleq	43
12.3	mod_maths Module Reference	43
12.3.1	Detailed Description	44
12.3.2	Description	44
12.3.3	Function/Subroutine Documentation	44

12.3.3.1	math_machine_eps	44
12.3.4	Variable Documentation	45
12.3.4.1	math_what_exception	45
12.4	mod_matrix Module Reference	45
12.4.1	Detailed Description	50
12.4.2	Description	51
12.4.3	Function/Subroutine Documentation	51
12.4.3.1	m_add	51
12.4.3.2	m_affect	51
12.4.3.3	m_bidiag_low	52
12.4.3.4	m_bidiag_up	52
12.4.3.5	m_cond	53
12.4.3.6	m_decompCholesky	53
12.4.3.7	m_decompLU	54
12.4.3.8	m_decompLU_m	54
12.4.3.9	m_decompQR	55
12.4.3.10	m_decompQR_GramSchmidt	55
12.4.3.11	m_decompQR_GramSchmidt_Reortho	56
12.4.3.12	m_decompQR_HouseHolder	57
12.4.3.13	m_decompsvd	57
12.4.3.14	m_decompsvd_s	58
12.4.3.15	m_destruct	59
12.4.3.16	m_destruct_lu	59
12.4.3.17	m_destruct_m_and_p	60
12.4.3.18	m_destruct_qr	60
12.4.3.19	m_destruct_t_eig	60
12.4.3.20	m_destruct_t_poweig	61
12.4.3.21	m_destruct_t_svd	61
12.4.3.22	m_det	62
12.4.3.23	m_det_chol	62
12.4.3.24	m_det_gaussj	63
12.4.3.25	m_det_lu	63
12.4.3.26	m_det_lu_all	64
12.4.3.27	m_diag	64
12.4.3.28	m_div_scalar	65
12.4.3.29	m_eig_deflation	65

12.4.3.30 <code>m_eig_qr</code>	66
12.4.3.31 <code>m_extract</code>	67
12.4.3.32 <code>m_get</code>	67
12.4.3.33 <code>m_get_m</code>	68
12.4.3.34 <code>m_getCol</code>	68
12.4.3.35 <code>m_getRow</code>	69
12.4.3.36 <code>m_getSize</code>	69
12.4.3.37 <code>m_getSizeCols</code>	69
12.4.3.38 <code>m_getSizeRows</code>	70
12.4.3.39 <code>m_identity</code>	70
12.4.3.40 <code>m_init</code>	71
12.4.3.41 <code>m_init_fromfile</code>	71
12.4.3.42 <code>m_inverse_gaussj</code>	72
12.4.3.43 <code>m_isEqual</code>	72
12.4.3.44 <code>m_isEqual_scalar</code>	73
12.4.3.45 <code>m_isSymmetric</code>	73
12.4.3.46 <code>m_matrixTOvector</code>	73
12.4.3.47 <code>m_max</code>	74
12.4.3.48 <code>m_maxCol</code>	74
12.4.3.49 <code>m_maxRow</code>	75
12.4.3.50 <code>m_min</code>	75
12.4.3.51 <code>m_minCol</code>	76
12.4.3.52 <code>m_minit_value</code>	76
12.4.3.53 <code>m_minRow</code>	76
12.4.3.54 <code>m_minus</code>	77
12.4.3.55 <code>m_nbnegative</code>	77
12.4.3.56 <code>m_nbnegativeCol</code>	78
12.4.3.57 <code>m_nbnegativeRow</code>	78
12.4.3.58 <code>m_nbpositive</code>	78
12.4.3.59 <code>m_nbpositiveCol</code>	79
12.4.3.60 <code>m_nbpositiveRow</code>	79
12.4.3.61 <code>m_nbzeros</code>	80
12.4.3.62 <code>m_nbzerosCol</code>	80
12.4.3.63 <code>m_nbzerosRow</code>	80
12.4.3.64 <code>m_norm</code>	81
12.4.3.65 <code>m_permut</code>	81

12.4.3.66	m_permut_col	82
12.4.3.67	m_pinv	82
12.4.3.68	m_pow_eig	83
12.4.3.69	m_print	84
12.4.3.70	m_print_lu_tofile	84
12.4.3.71	m_print_m_and_p_tofile	85
12.4.3.72	m_print_qr_tofile	85
12.4.3.73	m_print_tofile	86
12.4.3.74	m_prod_mat	86
12.4.3.75	m_prod_scalar1	87
12.4.3.76	m_prod_scalar2	87
12.4.3.77	m_prod_vec1	88
12.4.3.78	m_prod_vec2	88
12.4.3.79	m_prod_vec_c	89
12.4.3.80	m_pseudoinv_chol	89
12.4.3.81	m_pseudoinv_svd	90
12.4.3.82	m_rank	90
12.4.3.83	m_rank_gaussj	91
12.4.3.84	m_rank_svd	91
12.4.3.85	m_resize	92
12.4.3.86	m_set	92
12.4.3.87	m_setsub	93
12.4.3.88	m_sum	94
12.4.3.89	m_sumCol	94
12.4.3.90	m_sumRow	94
12.4.3.91	m_trace	95
12.4.3.92	m_trans	95
12.4.3.93	m_tril	95
12.4.3.94	m_triu	96
12.4.3.95	m_zeros	96
12.4.3.96	mc_bidiag_low	97
12.4.3.97	mc_bidiag_up	97
12.4.3.98	mc_diag	98
12.4.3.99	mc_diagDominant	98
12.4.3.100	mc_diagDominantSymmetric	98
12.4.3.101	mc_random	99

12.4.3.102mc_tridiag	99
12.4.4 Variable Documentation	100
12.4.4.1 frobenius	100
12.4.4.2 m_what_exception	100
12.5 mod_times Module Reference	100
12.5.1 Detailed Description	101
12.5.2 Description	101
12.5.3 Function/Subroutine Documentation	101
12.5.3.1 ft_begin	101
12.5.3.2 ft_create_time	102
12.5.3.3 ft_create_time_copy	102
12.5.3.4 ft_create_time_init	102
12.5.3.5 ft_end	103
12.5.3.6 ft_gettimeofday	103
12.5.3.7 ft_print	104
12.5.3.8 ft_reset	104
12.6 mod_utility Module Reference	104
12.6.1 Detailed Description	105
12.6.2 Description	105
12.6.3 Function/Subroutine Documentation	105
12.6.3.1 u_is_exist_file	105
12.6.3.2 u_nblne	106
12.6.4 Variable Documentation	106
12.6.4.1 u_what_exception	106
12.7 mod_vector Module Reference	106
12.7.1 Detailed Description	109
12.7.2 Description	109
12.7.3 Function/Subroutine Documentation	109
12.7.3.1 v_abs	109
12.7.3.2 v_add	110
12.7.3.3 v_add_val_end	110
12.7.3.4 v_affect	111
12.7.3.5 v_axpby	111
12.7.3.6 v_cross	111
12.7.3.7 v_destruct	112
12.7.3.8 v_div_scalar	112

12.7.3.9	<code>v_dot</code>	113
12.7.3.10	<code>v_extract</code>	113
12.7.3.11	<code>v_get</code>	113
12.7.3.12	<code>v_get_v</code>	114
12.7.3.13	<code>v_init</code>	114
12.7.3.14	<code>v_init_fromfile</code>	115
12.7.3.15	<code>v_init_value</code>	115
12.7.3.16	<code>v_inverse</code>	116
12.7.3.17	<code>v_isEqual</code>	116
12.7.3.18	<code>v_isEqual_scalar</code>	116
12.7.3.19	<code>v_length</code>	117
12.7.3.20	<code>v_max</code>	117
12.7.3.21	<code>v_min</code>	118
12.7.3.22	<code>v_minus</code>	118
12.7.3.23	<code>v_nbnegative</code>	119
12.7.3.24	<code>v_nbpositive</code>	119
12.7.3.25	<code>v_nbzeros</code>	120
12.7.3.26	<code>v_norm</code>	120
12.7.3.27	<code>v_normalize</code>	120
12.7.3.28	<code>v_ones</code>	121
12.7.3.29	<code>v_print</code>	121
12.7.3.30	<code>v_print_c</code>	122
12.7.3.31	<code>v_print_c_tofile</code>	122
12.7.3.32	<code>v_print_tofile</code>	123
12.7.3.33	<code>v_prod</code>	123
12.7.3.34	<code>v_prod_scalar1</code>	124
12.7.3.35	<code>v_prod_scalar2</code>	124
12.7.3.36	<code>v_prod_vec</code>	125
12.7.3.37	<code>v_resize</code>	125
12.7.3.38	<code>v_set</code>	125
12.7.3.39	<code>v_size</code>	126
12.7.3.40	<code>v_sqrLength</code>	126
12.7.3.41	<code>v_sum</code>	127
12.7.3.42	<code>v_zeros</code>	127
12.7.3.43	<code>vc_random</code>	128
12.7.4	Variable Documentation	128

12.7.4.1	infty	128
12.7.4.2	v_what_exception	128
13	Data Type Documentation	129
13.1	mod_vector::abs Interface Reference	129
13.1.1	Detailed Description	129
13.1.2	Member Function/Subroutine Documentation	129
13.1.2.1	v_abs	129
13.2	mod_vector::add Interface Reference	130
13.2.1	Detailed Description	130
13.2.2	Member Function/Subroutine Documentation	130
13.2.2.1	v_add_val_end	130
13.3	mod_matrix::assignment(=) Interface Reference	130
13.3.1	Detailed Description	130
13.3.2	Member Function/Subroutine Documentation	131
13.3.2.1	m_affect	131
13.4	mod_vector::assignment(=) Interface Reference	131
13.4.1	Detailed Description	131
13.4.2	Member Function/Subroutine Documentation	132
13.4.2.1	v_affect	132
13.5	mod_matrix::chol Interface Reference	132
13.5.1	Detailed Description	132
13.5.2	Member Function/Subroutine Documentation	132
13.5.2.1	m_decompCholesky	132
13.6	mod_linear_equation::destruct Interface Reference	132
13.6.1	Detailed Description	133
13.6.2	Member Function/Subroutine Documentation	133
13.6.2.1	m_destruct_soleq	133
13.7	mod_matrix::destruct Interface Reference	133
13.7.1	Detailed Description	134
13.7.2	Member Function/Subroutine Documentation	134
13.7.2.1	m_destruct	134
13.7.2.2	m_destruct_lu	134
13.7.2.3	m_destruct_m_and_p	134
13.7.2.4	m_destruct_qr	134
13.7.2.5	m_destruct_t_eig	134
13.7.2.6	m_destruct_t_poweig	134

13.7.2.7	m_destruct_t_svd	134
13.8	mod_vector::destruct Interface Reference	135
13.8.1	Detailed Description	135
13.8.2	Member Function/Subroutine Documentation	135
13.8.2.1	v_destruct	135
13.9	mod_matrix::det Interface Reference	135
13.9.1	Detailed Description	135
13.9.2	Member Function/Subroutine Documentation	136
13.9.2.1	m_det	136
13.10	mod_matrix::diag Interface Reference	136
13.10.1	Detailed Description	136
13.10.2	Member Function/Subroutine Documentation	136
13.10.2.1	m_diag	136
13.11	mod_vector::dot Interface Reference	137
13.11.1	Detailed Description	137
13.11.2	Member Function/Subroutine Documentation	137
13.11.2.1	v_dot	137
13.12	mod_matrix::get Interface Reference	137
13.12.1	Detailed Description	138
13.12.2	Member Function/Subroutine Documentation	138
13.12.2.1	m_get	138
13.12.2.2	m_get_m	138
13.12.2.3	m_getRow	138
13.13	mod_vector::get Interface Reference	138
13.13.1	Detailed Description	139
13.13.2	Member Function/Subroutine Documentation	139
13.13.2.1	v_get	139
13.13.2.2	v_get_v	139
13.14	mod_linear_equation::ilinsolve Interface Reference	139
13.14.1	Detailed Description	139
13.14.2	Member Function/Subroutine Documentation	140
13.14.2.1	leq_solve_iter	140
13.15	mod_vector::init Interface Reference	140
13.15.1	Detailed Description	140
13.15.2	Member Function/Subroutine Documentation	140
13.15.2.1	v_init	140

13.15.2.2 v_init_fromfile	141
13.16mod_matrix::init Interface Reference	141
13.16.1 Detailed Description	141
13.16.2 Member Function/Subroutine Documentation	141
13.16.2.1 m_init	141
13.16.2.2 m_init_fromfile	141
13.17mod_matrix::inv Interface Reference	142
13.17.1 Detailed Description	142
13.17.2 Member Function/Subroutine Documentation	142
13.17.2.1 m_inverse_gaussj	142
13.18mod_linear_equation::leq_getBlock Interface Reference	142
13.18.1 Detailed Description	142
13.18.2 Member Function/Subroutine Documentation	143
13.18.2.1 gsb_getBlock_m	143
13.18.2.2 leq_getBlock_v	143
13.19mod_linear_equation::linsolve Interface Reference	143
13.19.1 Detailed Description	143
13.19.2 Member Function/Subroutine Documentation	144
13.19.2.1 leq_solve	144
13.20mod_matrix::lu Interface Reference	144
13.20.1 Detailed Description	144
13.20.2 Member Function/Subroutine Documentation	145
13.20.2.1 m_decompLU	145
13.21mod_matrix::m_lu Interface Reference	145
13.21.1 Detailed Description	145
13.21.2 Member Function/Subroutine Documentation	145
13.21.2.1 m_decompLU_m	145
13.22mod_matrix::matrix Type Reference	145
13.22.1 Detailed Description	146
13.22.2 Member Data Documentation	146
13.22.2.1 cols	146
13.22.2.2 container	146
13.22.2.3 is_allocate	146
13.22.2.4 matrix	146
13.22.2.5 ptr_container	147
13.22.2.6 rows	147

13.23mod_vector::max Interface Reference	147
13.23.1 Detailed Description	147
13.23.2 Member Function/Subroutine Documentation	147
13.23.2.1 v_max	147
13.24mod_matrix::max Interface Reference	147
13.24.1 Detailed Description	148
13.24.2 Member Function/Subroutine Documentation	148
13.24.2.1 m_max	148
13.25mod_vector::min Interface Reference	148
13.25.1 Detailed Description	148
13.25.2 Member Function/Subroutine Documentation	149
13.25.2.1 v_min	149
13.26mod_matrix::min Interface Reference	149
13.26.1 Detailed Description	149
13.26.2 Member Function/Subroutine Documentation	149
13.26.2.1 m_min	149
13.27mod_vector::norm Interface Reference	150
13.27.1 Detailed Description	150
13.27.2 Member Function/Subroutine Documentation	150
13.27.2.1 v_norm	150
13.28mod_matrix::norm Interface Reference	150
13.28.1 Detailed Description	150
13.28.2 Member Function/Subroutine Documentation	151
13.28.2.1 m_norm	151
13.29mod_vector::operator(*) Interface Reference	151
13.29.1 Detailed Description	151
13.29.2 Member Function/Subroutine Documentation	152
13.29.2.1 v_prod_scalar1	152
13.29.2.2 v_prod_scalar2	152
13.29.2.3 v_prod_vec	152
13.30mod_matrix::operator(*) Interface Reference	152
13.30.1 Detailed Description	152
13.30.2 Member Function/Subroutine Documentation	153
13.30.2.1 m_prod_mat	153
13.30.2.2 m_prod_scalar1	153
13.30.2.3 m_prod_scalar2	153

13.30.2.4 m_prod_vec2	153
13.30.2.5 m_prod_vec_c	153
13.31 mod_matrix::operator(+) Interface Reference	153
13.31.1 Detailed Description	153
13.31.2 Member Function/Subroutine Documentation	154
13.31.2.1 m_add	154
13.32 mod_vector::operator(+) Interface Reference	154
13.32.1 Detailed Description	154
13.32.2 Member Function/Subroutine Documentation	154
13.32.2.1 v_add	154
13.33 mod_matrix::operator(-) Interface Reference	155
13.33.1 Detailed Description	155
13.33.2 Member Function/Subroutine Documentation	155
13.33.2.1 m_minus	155
13.34 mod_vector::operator(-) Interface Reference	155
13.34.1 Detailed Description	155
13.34.2 Member Function/Subroutine Documentation	156
13.34.2.1 v_minus	156
13.35 mod_matrix::operator(.cond.) Interface Reference	156
13.35.1 Detailed Description	156
13.35.2 Member Function/Subroutine Documentation	157
13.35.2.1 m_cond	157
13.36 mod_vector::operator(.cross.) Interface Reference	157
13.36.1 Detailed Description	157
13.36.2 Member Function/Subroutine Documentation	157
13.36.2.1 v_cross	157
13.37 mod_matrix::operator(.det.) Interface Reference	157
13.37.1 Detailed Description	158
13.37.2 Member Function/Subroutine Documentation	158
13.37.2.1 m_det_gaussj	158
13.38 mod_vector::operator(.dot.) Interface Reference	158
13.38.1 Detailed Description	158
13.38.2 Member Function/Subroutine Documentation	159
13.38.2.1 v_dot	159
13.39 mod_matrix::operator(.inv.) Interface Reference	159
13.39.1 Detailed Description	159

13.39.2 Member Function/Subroutine Documentation	159
13.39.2.1 m_inverse_gaussj	159
13.40mod_vector::operator(.inv.) Interface Reference	160
13.40.1 Detailed Description	160
13.40.2 Member Function/Subroutine Documentation	160
13.40.2.1 v_inverse	160
13.41mod_vector::operator(.len.) Interface Reference	160
13.41.1 Detailed Description	160
13.41.2 Member Function/Subroutine Documentation	161
13.41.2.1 v_size	161
13.42mod_vector::operator(.norm.) Interface Reference	161
13.42.1 Detailed Description	161
13.42.2 Member Function/Subroutine Documentation	161
13.42.2.1 v_length	161
13.43mod_matrix::operator(.rank.) Interface Reference	162
13.43.1 Detailed Description	162
13.43.2 Member Function/Subroutine Documentation	162
13.43.2.1 m_rank_gaussj	162
13.44mod_vector::operator(.sqnorm.) Interface Reference	162
13.44.1 Detailed Description	162
13.44.2 Member Function/Subroutine Documentation	163
13.44.2.1 v_sqrLength	163
13.45mod_matrix::operator(.tr.) Interface Reference	163
13.45.1 Detailed Description	163
13.45.2 Member Function/Subroutine Documentation	163
13.45.2.1 m_trans	163
13.46mod_matrix::operator(/) Interface Reference	164
13.46.1 Detailed Description	164
13.46.2 Member Function/Subroutine Documentation	164
13.46.2.1 m_div_scalar	164
13.47mod_vector::operator(/) Interface Reference	164
13.47.1 Detailed Description	164
13.47.2 Member Function/Subroutine Documentation	165
13.47.2.1 v_div_scalar	165
13.48mod_matrix::operator(==) Interface Reference	165
13.48.1 Detailed Description	165

13.48.2 Member Function/Subroutine Documentation	166
13.48.2.1 m_isEqual	166
13.48.2.2 m_isEqual_scalar	166
13.49 mod_vector::operator(==) Interface Reference	166
13.49.1 Detailed Description	166
13.49.2 Member Function/Subroutine Documentation	166
13.49.2.1 v_isEqual	166
13.49.2.2 v_isEqual_scalar	167
13.50 mod_matrix::pinv Interface Reference	167
13.50.1 Detailed Description	167
13.50.2 Member Function/Subroutine Documentation	167
13.50.2.1 m_pinv	167
13.51 mod_matrix::print Interface Reference	167
13.51.1 Detailed Description	168
13.51.2 Member Function/Subroutine Documentation	168
13.51.2.1 m_print	168
13.51.2.2 m_print_lu_tofile	168
13.51.2.3 m_print_tofile	168
13.52 mod_vector::print Interface Reference	168
13.52.1 Detailed Description	169
13.52.2 Member Function/Subroutine Documentation	169
13.52.2.1 v_print	169
13.52.2.2 v_print_tofile	169
13.53 mod_matrix::qr Interface Reference	169
13.53.1 Detailed Description	170
13.53.2 Member Function/Subroutine Documentation	170
13.53.2.1 m_decompQR	170
13.54 mod_vector::random Interface Reference	170
13.54.1 Detailed Description	170
13.54.2 Member Function/Subroutine Documentation	171
13.54.2.1 vc_random	171
13.55 mod_matrix::random Interface Reference	171
13.55.1 Detailed Description	171
13.55.2 Member Function/Subroutine Documentation	171
13.55.2.1 mc_random	171
13.56 mod_matrix::rank Interface Reference	171

13.56.1 Detailed Description	172
13.56.2 Member Function/Subroutine Documentation	172
13.56.2.1 m_rank	172
13.57mod_vector::set Interface Reference	172
13.57.1 Detailed Description	172
13.57.2 Member Function/Subroutine Documentation	173
13.57.2.1 v_set	173
13.58mod_matrix::set Interface Reference	173
13.58.1 Detailed Description	173
13.58.2 Member Function/Subroutine Documentation	173
13.58.2.1 m_set	173
13.59mod_matrix::spec Interface Reference	174
13.59.1 Detailed Description	174
13.59.2 Member Function/Subroutine Documentation	174
13.59.2.1 m_eig_qr	174
13.60mod_vector::sqnorm Interface Reference	174
13.60.1 Detailed Description	175
13.60.2 Member Function/Subroutine Documentation	175
13.60.2.1 v_sqrLength	175
13.61mod_vector::sum Interface Reference	175
13.61.1 Detailed Description	175
13.61.2 Member Function/Subroutine Documentation	176
13.61.2.1 v_sum	176
13.62mod_matrix::sum Interface Reference	176
13.62.1 Detailed Description	176
13.62.2 Member Function/Subroutine Documentation	176
13.62.2.1 m_sum	176
13.63mod_matrix::svd Interface Reference	176
13.63.1 Detailed Description	177
13.63.2 Member Function/Subroutine Documentation	177
13.63.2.1 m_decompsvd	177
13.64mod_matrix::t_eig Type Reference	177
13.64.1 Detailed Description	178
13.64.2 Member Data Documentation	179
13.64.2.1 m_eigvectors	179
13.64.2.2 v_eigvalues	179

13.65mod_matrix::t_lu Type Reference	179
13.65.1 Detailed Description	180
13.65.2 Member Data Documentation	181
13.65.2.1 L	181
13.65.2.2 P	181
13.65.2.3 swops	181
13.65.2.4 U	181
13.66mod_matrix::t_m_and_p Type Reference	181
13.66.1 Detailed Description	182
13.66.2 Member Data Documentation	183
13.66.2.1 M	183
13.66.2.2 P	183
13.66.2.3 swops	183
13.67mod_matrix::t_poweig Type Reference	183
13.67.1 Detailed Description	184
13.67.2 Member Data Documentation	185
13.67.2.1 iter	185
13.67.2.2 lambda	185
13.67.2.3 v_err	185
13.67.2.4 v_lambda	185
13.68mod_matrix::t_qr Type Reference	185
13.68.1 Detailed Description	186
13.68.2 Member Data Documentation	187
13.68.2.1 P	187
13.68.2.2 Q	187
13.68.2.3 R	187
13.68.2.4 swops	187
13.69mod_linear_equation::t_soleq Type Reference	187
13.69.1 Detailed Description	188
13.69.2 Member Data Documentation	189
13.69.2.1 iter	189
13.69.2.2 v_err	189
13.69.2.3 v_sol	189
13.70mod_matrix::t_svd Type Reference	189
13.70.1 Detailed Description	190
13.70.2 Member Data Documentation	191

13.70.2.1 S	191
13.70.2.2 U	191
13.70.2.3 V	191
13.71mod_times::t_time Type Reference	191
13.71.1 Detailed Description	193
13.71.2 Member Data Documentation	193
13.71.2.1 t_elapsed	193
13.71.2.2 t_end	193
13.71.2.3 t_icount	193
13.71.2.4 t_inbcalls	193
13.71.2.5 t_ioffset	194
13.71.2.6 t_name	194
13.71.2.7 t_scale	194
13.71.2.8 t_start	194
13.71.2.9 t_t	194
13.71.2.10 t1	194
13.71.2.11 t2	194
13.71.2.12 unit	194
13.72mod_times::t_timeval Type Reference	194
13.72.1 Detailed Description	195
13.72.2 Member Data Documentation	195
13.72.2.1 tv_msec	195
13.72.2.2 tv_nsec	195
13.72.2.3 tv_sec	195
13.72.2.4 tv_usec	195
13.73mod_matrix::tril Interface Reference	195
13.73.1 Detailed Description	196
13.73.2 Member Function/Subroutine Documentation	196
13.73.2.1 m_tril	196
13.74mod_matrix::triu Interface Reference	196
13.74.1 Detailed Description	196
13.74.2 Member Function/Subroutine Documentation	197
13.74.2.1 m_triu	197
13.75mod_exception::type_exception Type Reference	197
13.75.1 Detailed Description	197
13.75.2 Member Data Documentation	198

13.75.2.1	e_level	198
13.75.2.2	e_number	198
13.75.2.3	e_what	198
13.76	mod_vector::vector Type Reference	198
13.76.1	Detailed Description	198
13.76.2	Member Data Documentation	199
13.76.2.1	is_allocate	199
13.76.2.2	ptr_container	199
13.76.2.3	size	199
14	File Documentation	201
14.1	exception.f90 File Reference	201
14.2	fml_constants.h File Reference	202
14.2.1	Define Documentation	203
14.2.1.1	DEBUG_EXCEPTION	203
14.2.1.2	DEBUGPRINT	203
14.2.1.3	DEBUGPRINT_MPI	204
14.2.1.4	len_what_exception	204
14.2.1.5	MASTER_ID	204
14.2.1.6	mpi_type_precision	204
14.2.1.7	OMP_NUM_THREADS	204
14.2.1.8	p_abs	204
14.2.1.9	p_cast	204
14.2.1.10	p_dble	204
14.2.1.11	p_eps	204
14.2.1.12	p_fmt_file	204
14.2.1.13	p_iter_eps	204
14.2.1.14	p_mun	205
14.2.1.15	p_notcast	205
14.2.1.16	p_sngl	205
14.2.1.17	p_sqrt	205
14.2.1.18	p_un	205
14.2.1.19	stop_alloc	205
14.2.1.20	stop_arith	205
14.2.1.21	stop_array_compatible	205
14.2.1.22	stop_array_diag_compatible	205
14.2.1.23	stop_array_diverge	205

14.2.1.24	stop_array_factorisation	205
14.2.1.25	stop_array_indice_exceed	206
14.2.1.26	stop_array_positive	206
14.2.1.27	stop_array_singular	206
14.2.1.28	stop_array_symmetric	206
14.2.1.29	stop_dealloc	206
14.2.1.30	stop_div0	206
14.2.1.31	stop_err	206
14.2.1.32	stop_method_exist	206
14.2.1.33	stop_open_file	206
14.2.1.34	stop_overflow	206
14.2.1.35	stop_sqrt	206
14.2.1.36	stop_std	207
14.2.1.37	stop_tst	207
14.2.1.38	stop_write_file	207
14.2.1.39	type_precision	207
14.2.1.40	use_type	207
14.3	fml_debug.h File Reference	207
14.4	fml_timing.h File Reference	208
14.4.1	Define Documentation	208
14.4.1.1	CLOCKS_PER_SEC	208
14.4.1.2	dd_nbCalls	208
14.4.1.3	dd_OffSet	208
14.5	info.f90 File Reference	208
14.5.1	Function Documentation	208
14.5.1.1	info	208
14.6	linear_equation.f90 File Reference	209
14.6.1	Detailed Description	210
14.7	maths.f90 File Reference	210
14.7.1	Detailed Description	211
14.8	matrix.f90 File Reference	211
14.8.1	Detailed Description	217
14.9	times.f90 File Reference	217
14.9.1	Detailed Description	219
14.10	utility.f90 File Reference	219
14.10.1	Detailed Description	220

14.11 vector.f90 File Reference	220
14.11.1 Detailed Description	224
15 Example Documentation	225
15.1 leq_exception.f90	225
15.2 leq_solve.f90	226
15.3 leq_solve.f90	227
15.4 leq_solve_times.f90	229
15.5 matrix_analysis.f90	231
15.6 matrix_analysis2.f90	232
15.7 matrix_cholesky.f90	233
15.8 matrix_eigenvalvect.f90	234
15.9 matrix_exception.f90	235
15.10 matrix_init.f90	236
15.11 matrix_lu.f90	236
15.12 matrix_manip.f90	237
15.13 matrix_qr.f90	239
15.14 matrix_svd.f90	240
15.15 vector_exception.f90	241
15.16 vector_init.f90	242
15.17 vector_manip.f90	242

Chapter 1

Fortran Mathematics Library



1.1 Introduction

This code has been performed by Abal-Kassim Cheik Ahamed, It is freed at Ecole Internationale des Sciences du Traitement de l'Information (EISTI) and Universit of Cergy-Pontoise (UCP) under licence CeCILL.

This software is created by Abal-Kassim Cheik Ahamed for a school project of his last year of engineer school (Scientific Computing Option)

Copyright or (c) or Copr. Abal-Kassim Cheik Ahamed (student of EISTI<www.eisti.fr> and University of Cergy<www.u-cergy.fr> (May 01 2011)

akcheik@gmail.com

This software is a computer program whose purpose is to fortran mathematics library and some solve linear

equation in high performance computing.

This software is governed by the CeCILL license under French law and abiding by the rules of distribution of free software. You can use, modify and/ or redistribute the software under the terms of the CeCILL license as circulated by CEA, CNRS and INRIA at the following URL "<http://www.cecill.info>".

As a counterpart to the access to the source code and rights to copy, modify and redistribute granted by the license, users are provided only with a limited warranty and the software's author, the holder of the economic rights, and the successive licensors have only limited liability.

In this respect, the user's attention is drawn to the risks associated with loading, using, modifying and/or developing or reproducing the software by the user in light of its specific status of free software, that may mean that it is complicated to manipulate, and that also therefore means that it is reserved for developers and experienced professionals having in-depth computer knowledge. Users are therefore encouraged to load and test the software's suitability as regards their requirements in conditions enabling the security of their systems and/or data to be ensured and, more generally, to use and operate it in the same conditions as regards security.

The fact that you are presently reading this means that you have had knowledge of the CeCILL license and that you accept its terms.

Bibliography :

* Youcef Saad : >> <http://www-users.cs.umn.edu/~saad/>

Iterative methods for sparse linear systems (2nd edition)

* wikipédia : >> http://fr.wikipedia.org/wiki/%C3%89limination_de_Gauss-Jordan

Élimination de Gauss-Jordan

>> http://fr.wikipedia.org/wiki/D%C3%A9composition_QR

Décomposition QR

[Files Tree](#)

[Notes on building](#)

[Notes on make help](#)

[Run methods](#)

Source code can be obtained from <akcheik@gmail.com>

Author

Abal-Kassim Cheik Ahamed , akcheik@gmail.com (EISTI, UCP)

Version

1.0

Date

13th of March, 2011

Remarks

Chapter 2

Files Tree

```
----->.\n/ :
** common: common files
** examples: examples code
** images: images
** include: header files
** licence: licence file
** src: source files
** CMakeLists.txt: main CMakeLists.txt for cmake
** COPYING: licence resume
** Doxyfile.in: doxygen file config
** ReadMe.txt: readme file

----->./common :
** cmake_uninstall.cmake.in: for uninstall the program
** CPack.cmake: from cmake Modules
** Fortran.cmake: fortran optimization config
** UseBoostConfig.cmake: boost config
** UseConfig.cmake: general configuration
** UseCPackConfig.cmake: cpack personal config
** UseDoxygen.cmake: from cmake Modules
** UseDoxygenConfig.cmake: personal doxygen config
** UseMpiConfig.cmake: personal mpi config
** UseOpenMPConfig.cmake: personal openmpi config
** UsePlatform.cmake:
** version.cmake: version config

----->./examples :
** examples/leq
**** CMakeLists.txt:
**** leq_exception.f90:
**** leq_isolve.f90:
**** leq_solve.f90:
**** main_leq.f90:
** examples/matrix
**** CMakeLists.txt:
**** matrix_analysis.f90:
**** matrix_analysis2.f90:
**** matrix_cholesky.f90:
**** matrix_eigenvalvect.f90:
**** matrix_exception.f90:
**** matrix_init.f90:
**** matrix_lu.f90:
**** matrix_manip.f90:
**** matrix_qr.f90:
**** matrix_svd.f90:
```

```
** examples/vector
**** CMakeLists.txt:
**** vector_exception.f90:
**** vector_init.f90:
**** vector_manip.f90:
** examples/CMakeLists.txt

----->./include (header files)
**CMakeLists.txt:
**fml_constants.h:
**fml_debug.h:
**fml_timing.h:

\verbatimim
----->./licence (licence files)
**Licence_CeCILL_V2-en.txt: licence file (english version)
**Licence_CeCILL_V2-fr.txt: licence file (french version)

----->./src (source files)
** CMakeLists.txt:
** exception.f90:
** info.f90:
** linear_equation.f90:
** main.f90:
** maths.f90:
** matrix.f90:
** times.f90:
** utility.f90:
** vector.f90:
```

Chapter 3

Notes on building

3.1 Description

Using "Fortran Mathematics Library" requires fortran compiler (gfortran, ifort), cmake build system.

```
All options except BUILD_SHARED_LIBS are setting OFF
* FML_USE_MPI: enable/disable MPI (ON or OFF)
* FML_USE_OPENMP: enable/disable OPENMP (ON or OFF)
* FML_USE_TIMING: enable/disable TIMING (ON or OFF)
* FML_USE_IFORT: enable/disable Intel ifort non-commercial Compiler (ON or OFF)
* FML_DEBUG_EXCEPTION: "enable/disable debug exception (ON or OFF)
* FML_FAST: enable/disable make examples (ON or OFF)
* FML_DEBUG: enable/disable make examples (ON or OFF)
* FML_MAKE_EXAMPLES: enable/disable make examples (ON or OFF)
* FML_MAKE_TESTS: enable/disable make tests (by CTest)" (ON or OFF)
* FML_MAKE_LIBRARIES: enable/disable make libraries (ON or OFF)
* FML_INSTALL_DOC: set to OFF to skip build/install Documentation (ON or OFF)
* BUILD_SHARED_LIBS: set ON to build shared libraries (ON or OFF)
* FML_FORCE_64BIT: enable/disable debug (ON or OFF)
* FML_FORCE_WIN: enable/disable windows configuration (ON or OFF)
```

How compile this program :

```
*** Example 1: simple use (default INSTALL_PREFIX=/usr/lib)
akca@pc-eisti-school:$ mkdir ./bin
akca@pc-eisti-school:$ cd ./bin
akca@pc-eisti-school:$ cmake ../ (main directory, CMakeLists.txt must be on the previous directory)
akca@pc-eisti-school:$ make
```

```
*** Example 2: compile examples and change prefix directory
CMAKE_INSTALL_PREFIX is most often defined (-D) using a command line argument
when invoking CMake:
cmake <src-path> -D CMAKE_INSTALL_PREFIX=<install-path>
akca@pc-eisti-school:$ mkdir ./bin; mkdir ./fml
akca@pc-eisti-school:$ cd ./bin
akca@pc-eisti-school:$ cmake ../ -DFML_MAKE_EXAMPLES=ON -DCMAKE_INSTALL_PREFIX=../fml
akca@pc-eisti-school:$ make
akca@pc-eisti-school:$ make install
```

```
*** Example 3: make a package
akca@pc-eisti-school:$ mkdir ./bin; mkdir ./fml
akca@pc-eisti-school:$ cd ./bin
```

```
akca@pc-eisti-school:$ make package
akca@pc-eisti-school:$ make package_source (make a source package)
if create .deb package :
akca@pc-eisti-school:$ sudo dpkg -i fortran_mathematics_library-1.0.0-$ARCHI.deb
This will result in INSTALL_PREFIX
```

```
*** Example 4: If you install programs (make install)
***** bin/ : binary files
***** COPYING : licence files
***** include/ : header files
***** lib/ : library files
***** licence/ : licence directory
***** mod/ : module directory (for compilation without source files)
***** ReadMe.txt (documentation)
```

Chapter 4

Run methods

How compile execute program :

```
Example 1: assume that you have already compile the program (on ./bin directory)
akca@pc-eisti-school:$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:"PATH OF INSTALL_PREFIX"
akca@pc-eisti-school:$ cd ./bin
akca@pc-eisti-school:$ ./src/fml-info.x (more information about the program)
akca@pc-eisti-school:$ ./examples/vector/vector_init.x (if examples is created)
akca@pc-eisti-school:$ ./examples/matrix/matrix_init.x (if examples is created)
akca@pc-eisti-school:$ ./examples/leq/leq_solve.x (if examples is created)
```


Chapter 5

Notes on make help

5.1 Description

The following are some of the valid targets for this Makefile:

```
... all (the default if no target is provided)
... clean
... depend
... doc
... doxygen
... edit_cache
... install
... install/local
... install/strip
... list_install_components
... package
... package_source
... rebuild_cache
... uninstall
... exception
... fml
... fml-info.x
... leq
... maths
... matrix
... utility
... vector
... vector_exception.x
... vector_init.x
... vector_manip.x
... matrix_analysis.x
... matrix_analysis2.x
... matrix_cholesky.x
... matrix_eigenvalvect.x
... matrix_exception.x
... matrix_init.x
... matrix_lu.x
... matrix_manip.x
... matrix_qr.x
... matrix_svd.x
... leq_exception.x
... leq_solve.x
... leq_solve.x
```


Chapter 6

Todo List

Subprogram `mod_linear_equation::leq_Cholesky(m, v, is_permuted)` fill-permutation matrix ? re-build permutation matrix

Subprogram `mod_linear_equation::leq_getBlock_v(v, r_s, r_e)` verify why it's not work without res_pb declaration

Type `mod_linear_equation::linsolve` entrain de les faire

Subprogram `mod_matrix::m_cond(m)` verify avantage between fortran basic function and loop implementation
do with singular value ($=s[\min(m,n)-1]$)

Subprogram `mod_matrix::m_decompCholesky(m, is_permuted)` fill-permutation matrix ?

Subprogram `mod_matrix::m_decompQR(m, meth_qr, eps_gsorto, is_permuted)` implement Givens method

Subprogram `mod_matrix::m_decompQR_GramSchmidt(m, is_permuted)` rank? Q and R sizes

Subprogram `mod_matrix::m_det_chol(m, is_permuted)` check the determinant computing

Subprogram `mod_matrix::m_diag(m, i)` optimize the if conditions

Subprogram `mod_matrix::m_eig_deflation(m, v0, eps, iter_max)` do not work fine (after the second eigen value)

Subprogram `mod_matrix::m_matrixTOvector(m)` add exception

Subprogram `mod_matrix::m_norm(m, type_norm)` verify advantage between fortran basic function and loop implementation

Subprogram `mod_matrix::m_pow_eig(m, v0, eps, iter_max)` treat complex case

Subprogram `mod_matrix::m_prod_vec2(v, m)` parallel omp ? (ex, conjugate gradient)

Subprogram `mod_matrix::m_prod_vec_c(m, v)` parallel omp ? (ex, conjugate gradient)

Subprogram `mod_matrix::m_rank(m, tol_rank, meth_rk, eps_svd, iter_max, meth_qr, eps_gsorth, is_permuted)` implement Givens method

Subprogram `mod_matrix::m_rank_gaussj(m)` it's not working well (use rank_svd)

Subprogram `mod_vector::v_dot(v1, v2)` openmp is better ?

Chapter 7

Directory Hierarchy

7.1 Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

src 21

Chapter 8

Modules Index

8.1 Modules List

Here is a list of all modules with brief descriptions:

mod_exception (Module mod_exception)	23
mod_linear_equation (Module mod_linear_equation)	25
mod_maths (Module mod_maths)	43
mod_matrix (Module mod_matrix)	45
mod_times (Module mod_times)	100
mod_utility (Module mod_utility)	104
mod_vector (Module mod_vector)	106

Chapter 9

Data Type Index

9.1 Data Types List

Here are the data types with brief descriptions:

mod_vector::abs (Interface generic --> abs)	129
mod_vector::add (Interface generic --> add)	130
mod_matrix::assignment(=) (Brief interface assignment(=))	130
mod_vector::assignment(=) (Brief interface assignment(=))	131
mod_matrix::chol (Interface generic --> chol)	132
mod_linear_equation::destruct (Interface generic --> destruct linear_equation and derived type)	132
mod_matrix::destruct (Interface generic --> destruct matrix and derived type)	133
mod_vector::destruct (Interface generic --> destruct vector)	135
mod_matrix::det (Interface generic --> det)	135
mod_matrix::diag (Interface generic --> diag)	136
mod_vector::dot (Interface generic --> dot)	137
mod_matrix::get (Interface generic --> get)	137
mod_vector::get (Interface generic --> get)	138
mod_linear_equation::ilinsolve (Interface generic --> ilinsolve)	139
mod_vector::init (Interface generic --> init)	140
mod_matrix::init (Interface generic --> init)	141
mod_matrix::inv (Interface generic --> inv)	142
mod_linear_equation::leq_getBlock (Interface generic --> leq_getBlock)	142
mod_linear_equation::linsolve (Interface generic --> linsolve)	143
mod_matrix::lu (Interface generic --> lu)	144
mod_matrix::m_lu (Interface generic --> m_lu)	145
mod_matrix::matrix (Type matrix)	145
mod_vector::max (Interface generic --> max)	147
mod_matrix::max (Interface generic --> max)	147
mod_vector::min (Interface generic --> min)	148
mod_matrix::min (Interface generic --> min)	149
mod_vector::norm (Interface generic --> norm)	150
mod_matrix::norm (Interface generic --> norm)	150
mod_vector::operator(*) (Interface operator(*))	151
mod_matrix::operator(*) (Interface operator(*))	152
mod_matrix::operator(+) (Interface operator(+))	153
mod_vector::operator(+) (Interface operator(+))	154
mod_matrix::operator(-) (Interface operator(-))	155

<code>mod_vector::operator(-)</code> (Interface <code>operator(-)</code>)	155
<code>mod_matrix::operator(.cond.)</code> (Interface <code>operator(.cond.)</code>)	156
<code>mod_vector::operator(.cross.)</code> (Interface <code>operator(.cross.)</code>)	157
<code>mod_matrix::operator(.det.)</code> (Interface <code>operator(.det.)</code>)	157
<code>mod_vector::operator(.dot.)</code> (Interface <code>operator(.dot.)</code>)	158
<code>mod_matrix::operator(.inv.)</code> (Interface <code>operator(.inv.)</code>)	159
<code>mod_vector::operator(.inv.)</code> (Interface <code>operator(.inv.)</code>)	160
<code>mod_vector::operator(.len.)</code> (Interface <code>operator(.len.)</code>)	160
<code>mod_vector::operator(.norm.)</code> (Interface <code>operator(.norm.)</code>)	161
<code>mod_matrix::operator(.rank.)</code> (Interface <code>operator(.rank.)</code>)	162
<code>mod_vector::operator(.sqnorm.)</code> (Interface <code>operator(.sqnorm.)</code>)	162
<code>mod_matrix::operator(.tr.)</code> (Interface <code>operator(.tr.)</code>)	163
<code>mod_matrix::operator(/)</code> (Interface <code>operator(/)</code>)	164
<code>mod_vector::operator(/)</code> (Interface <code>operator(/)</code>)	164
<code>mod_matrix::operator(==)</code> (Interface <code>operator(==)</code>)	165
<code>mod_vector::operator(==)</code> (Interface <code>operator(==)</code>)	166
<code>mod_matrix::pinv</code> (Interface generic --> <code>pinv</code>)	167
<code>mod_matrix::print</code> (Interface generic --> <code>print</code>)	167
<code>mod_vector::print</code> (Interface generic --> <code>print</code>)	168
<code>mod_matrix::qr</code> (Interface generic --> <code>qr</code>)	169
<code>mod_vector::random</code> (Interface generic --> <code>random</code>)	170
<code>mod_matrix::random</code> (Interface generic --> <code>random</code>)	171
<code>mod_matrix::rank</code> (Interface generic --> <code>rank</code>)	171
<code>mod_vector::set</code> (Interface generic --> <code>set</code>)	172
<code>mod_matrix::set</code> (Interface generic --> <code>set</code>)	173
<code>mod_matrix::spec</code> (Interface generic --> <code>spec</code>)	174
<code>mod_vector::sqnorm</code> (Interface generic --> <code>sqnorm</code>)	174
<code>mod_vector::sum</code> (Interface generic --> <code>sum</code>)	175
<code>mod_matrix::sum</code> (Interface generic --> <code>sum</code>)	176
<code>mod_matrix::svd</code> (Interface generic --> <code>svd</code>)	176
<code>mod_matrix::t_eig</code> (Type <code>t_eig</code>)	177
<code>mod_matrix::t_lu</code> (Type <code>t_lu</code>)	179
<code>mod_matrix::t_m_and_p</code> (Type <code>t_m_and_p</code>)	181
<code>mod_matrix::t_poweig</code> (Type <code>t_poweig</code>)	183
<code>mod_matrix::t_qr</code> (Type <code>t_qr</code>)	185
<code>mod_linear_equation::t_oleq</code> (Type <code>t_oleq</code>)	187
<code>mod_matrix::t_svd</code> (Type <code>t_svd</code>)	189
<code>mod_times::t_time</code> (Type <code>t_time</code>)	191
<code>mod_times::t_timeval</code> (Type <code>t_timeval</code>)	194
<code>mod_matrix::tril</code> (Interface generic --> <code>tril</code>)	195
<code>mod_matrix::triu</code> (Interface generic --> <code>triu</code>)	196
<code>mod_exception::type_exception</code> (Type <code>type_exception</code>)	197
<code>mod_vector::vector</code> (Real*4 : type of precision : *real*4 for simple precision *real*8 for double precision)	198

Chapter 10

File Index

10.1 File List

Here is a list of all files with brief descriptions:

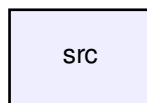
exception.f90	201
fml_constants.h	202
fml_debug.h	207
fml_timing.h	208
info.f90	208
linear_equation.f90 (Solve linear equation library)	209
maths.f90 (Utilities functions)	210
matrix.f90 (Exception)	211
times.f90 (Times)	217
utility.f90 (Utilities functions)	219
vector.f90 (Vector library)	220

Chapter 11

Directory Documentation

11.1 /home/akc/Desktop/projet_fortran/fml/src/ Directory Reference

Directory dependency graph for /home/akc/Desktop/projet_fortran/fml/src/:



Files

- file [exception.f90](#)
- file [info.f90](#)
- file [linear_equation.f90](#)
solve linear equation library
- file [maths.f90](#)
utilities functions
- file [matrix.f90](#)
exception
- file [times.f90](#)
times
- file [utility.f90](#)

utilities functions

- file [vector.f90](#)

vector library

Chapter 12

Module Documentation

12.1 mod_exception Module Reference

module [mod_exception](#)

Data Types

- type [type_exception](#)
type [type_exception](#)

Functions/Subroutines

- type([type_exception](#)) [e_error](#) (number_, what_, level_)
subroutine [e_print_err](#)(err_)
- subroutine [e_print_err](#) (err_)
subroutine [e_print_err](#)(err_)

12.1.1 Detailed Description



module [mod_exception](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Version

1.0

Date

12th of March, 2011

module for exception library

Remarks

depend : [fml_constants.h](#)

12.1.2 Description

...

12.1.3 Function/Subroutine Documentation**12.1.3.1** `type(type_exception) mod_exception::e_error (integer,intent(in) number_,
character(len= 100),intent(in) what_, integer,intent(in) level_)`

subroutine e_print_err(err_)

init a [type_exception](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

number_ : error number

level_ : error level

what_ : error signification

Date

12th of March, 2011

Remarks**Returns**

err_ : the error type

Attention

destructor and add the stop

Definition at line 49 of file exception.f90.

12.1.3.2 subroutine mod_exception::e_print_err (type(type_exception),intent(in) err_)

subroutine e_print_err(err_)

display the error

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

err_ : type(type_exception)

Date

12th of March, 2011

Remarks

Definition at line 69 of file exception.f90.

12.2 mod_linear_equation Module Reference

module [mod_linear_equation](#)

Data Types

- interface [destruct](#)
interface generic --> destruct linear_equation and derived type
- interface [ilinsolve](#)
interface generic --> ilinsolve
- interface [leq_getBlock](#)
interface generic --> [leq_getBlock](#)
- interface [linsolve](#)
interface generic --> linsolve
- type [t_soleq](#)
type [t_soleq](#)

Functions/Subroutines

- type(matrix) [gsb_getBlock_m](#) (m, r_s, r_e, c_s, c_e)
- integer, dimension(size_block) [gsb_index_blocks](#) (tab_block, size_block)
- type([t_soleq](#)) [leq_cg_elt](#) (m, v, v0, eps, iter_max, is_precond)

- type([vector](#)) [leq_Cholesky](#) (m, v, is_permuted)
- type([vector](#)) [leq_gaussJourdan](#) (m, v)
- type([t_soleq](#)) [leq_gaussSeidel_block](#) (m, v, v0, tab_block_i, size_block_i, eps, iter_max, is_precond)
- type([t_soleq](#)) [leq_gaussSeidel_elt](#) (m, v, v0, eps, iter_max, is_precond)
- type([vector](#)) [leq_getBlock_v](#) (v, r_s, r_e)
- type([t_soleq](#)) [leq_gradient_elt](#) (m, v, v0, eps, iter_max, is_precond)
- type([t_soleq](#)) [leq_jacobi_elt](#) (m, v, v0, eps, iter_max, is_precond)
- type([vector](#)) [leq_leastSquare](#) (m, v)
- type([vector](#)) [leq_leastSquare_QR](#) (m, v, meth_qr, eps_gsortho, is_permuted)
- type([vector](#)) [leq_lu](#) (m, v, is_permuted)
- type([vector](#)) [leq_precond_diag](#) (m, v)
- type([vector](#)) [leq_precond_sor](#) (m, v, v0, w_relax)
- type([vector](#)) [leq_pseudoinverse](#) (m, v, eps_svd, eps_chol, iter_max, meth_qr, eps_gsortho, meth_pinv)
- type([vector](#)) [leq_qr](#) (m, v, meth_qr, eps_gsortho, is_permuted)
- type([vector](#)) [leq_solve](#) (m, v, meth_solve, v0, eps_svd, eps_chol, iter_max, meth_qr, meth_pinv, eps_gsortho, is_permuted)
- type([t_soleq](#)) [leq_solve_iter](#) (m, v, meth_solve, v0, eps, iter_max, is_precond, w_relax, tab_block_i, size_block_i)
- type([t_soleq](#)) [leq_sor_elt](#) (m, v, v0, eps, iter_max, w_relax, is_precond)
- type([vector](#)) [leq_thomas](#) (m, v)
- type([vector](#)) [leq_tril](#) (m, v)
- type([vector](#)) [leq_triu](#) (m, v)
- subroutine [m_destruct_soleq](#) (soleq_)

12.2.1 Detailed Description

Linear Equa

module [mod_linear_equation](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Version

1.0

Date

12th of March, 2011

module for [mod_linear_equation](#) library

Remarks

depend : [fml_constants.h](#)

12.2.2 Description

This module `mod_linear_equation` defines some linear equation solve method.

See also

[mod_matrix](#)
[mod_vector](#)
[fml_constants.h](#)

12.2.3 Function/Subroutine Documentation

12.2.3.1 `type(matrix) mod_linear_equation::gsb_getBlock_m (type(matrix),intent(in) m,
integer,intent(in) r_s, integer,intent(in) r_e, integer,intent(in) c_s, integer,intent(in) c_e
)`

recover block

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
r_s : row start
r_e : row end
c_s : col start
c_e : col end

Date

13th of March, 2011

Remarks

extract block of *m* delimited by the previous parameters

Returns

res : type(matrix)

Exceptions

- + • *m* must be allocated
 - $1 \leq r_s \leq m\%rows$
 - $1 \leq r_e \leq m\%rows$
 - $1 \leq c_s \leq m\%cols$
 - $1 \leq c_e \leq m\%cols$

Definition at line 788 of file `linear_equation.f90`.

12.2.3.2 integer,dimension(size_block) mod_linear_equation::gsb_index_blocks (integer,dimension(size_block),intent(in) *tab_block*, integer,intent(in) *size_block*)

recover block indices from a vector contain size block

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

tab_block : type(vector)

size_block : size of *tab_block*

Date

13th of March, 2011

Remarks

Returns

res : type(vector)

Definition at line 755 of file linear_equation.f90.

12.2.3.3 type(t_soleq) mod_linear_equation::leq_cg_elt (type(matrix),intent(in) *m*, type(vector),intent(in) *v*, type(vector),intent(in),optional *v0*, real*4,intent(in),optional *eps*, integer,intent(in),optional *iter_max*, logical,intent(in),optional *is_precond*)

solve $mx=v$ by conjugate gradient

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

v0 : type(vector), initial vector (optional)

eps : real*4 , tolerance error (optional)

is_precond : if .true -> with preconditionnor (optional)

iter_max : integer, iteration max (optional)

Date

13th of March, 2011

Remarks

matrix must be hermitian (symmetric for real matrix)

Returns

res : type(t_soleq)

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 1117 of file linear_equation.f90.

12.2.3.4 type(vector) mod_linear_equation::leq_Cholesky (type(matrix),intent(in) m, type(vector),intent(in) v, logical,intent(in),optional is_permuted)

solve $mx=v$ by cholesky decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
v : type(vector)
is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks

matrix must be positive definite

Returns

res : type(vector)

See also

[math_machine_eps\(\)](#) : epsilon error

Todo

fill-permutation matrix ? rebuild permutation matrix

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 1652 of file linear_equation.f90.

12.2.3.5 `type(vector) mod_linear_equation::leq_gaussJourdan (type(matrix),intent(in) m, type(vector),intent(in) v)`

solve $mx=v$ by gauss jourdan

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

Date

12th of March, 2011

Remarks

Returns

res : type(vector)

See also

[math_machine_eps\(\)](#) : epsilon error

Exceptions

- + • *m* and *v* must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 114 of file linear_equation.f90.

12.2.3.6 `type(t_soleq) mod_linear_equation::leq_gaussSeidel_block (type(matrix),intent(in) m, type(vector),intent(in) v, type(vector),intent(in),optional v0, integer,dimension(:),intent(in),optional tab_block_i, integer,intent(in),optional size_block_i, real*4,intent(in),optional eps, integer,intent(in),optional iter_max, logical,intent(in),optional is_precond)`

solve $mx=v$ by gauss seidel block

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

v0 : type(vector), initiale vector

tab_block_i : block index
size_block_i : size block index
eps : real*4 , tolerance error (optional)
iter_max : integer, iteration max (optional)
is_precond : if .true -> with preconditionnor (optional)

Date

13th of March, 2011

Remarks**Returns**

res : type(t_soleq)

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 880 of file linear_equation.f90.

12.2.3.7 `type(t_soleq) mod_linear_equation::leq_gaussSeidel_elt (type(matrix),intent(in) m, type(vector),intent(in) v, type(vector),intent(in),optional v0, real*4,intent(in),optional eps, integer,intent(in),optional iter_max, logical,intent(in),optional is_precond)`

solve $mx=v$ by gauss seidel

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
v : type(vector)
v0 : type(vector), initial vector (optional)
eps : real*4 , tolerance error (optional)
iter_max : integer, iteration max (optional) (optional)
is_precond : if .true -> with preconditionnor (optional)

Date

13th of March, 2011

Remarks

Returns

res : type(t_soleq)

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 634 of file linear_equation.f90.

12.2.3.8 **type(vector) mod_linear_equation::leq_getBlock_v (type(vector),intent(in) v, integer,intent(in) r_s, integer,intent(in) r_e)**

recover block of vector column

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

r_s : row start

r_e : row end

Date

14th of March, 2011

Remarks**Returns**

res : type(matrix)

Todo

verify why it's not work without res_pb declaration

Exceptions

- + • v must be allocated
 - $1 \leq r_s \leq v\%size$
 - $1 \leq r_e \leq v\%size$

Definition at line 839 of file linear_equation.f90.

12.2.3.9 **type(t_soleq) mod_linear_equation::leq_gradient_elt (type(matrix),intent(in) m, type(vector),intent(in) v, type(vector),intent(in),optional v0, real*4,intent(in),optional eps, integer,intent(in),optional iter_max, logical,intent(in),optional is_precond)**

solve $mx=v$ by gradient method

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
v : type(vector)
v0 : type(vector), initial vector (optional)
eps : real*4 , tolerance error (optional)
iter_max : integer, iteration max (optional)
is_precond : if .true -> with preconditionnor (optional)

Date

13th of March, 2011

Remarks

matrix must be hermitian (symmetric for real matrix)

Returns

res : type(t_soleq)

Exceptions

- + • m and v must be allocated
 - *m*%rows = *v*%size
 - *m*%rows = *m*%cols (square matrix expected)

Definition at line 1256 of file linear_equation.f90.

12.2.3.10 type(t_soleq) mod_linear_equation::leq_jacobi_elt (type(matrix),intent(in) *m*,
 type(vector),intent(in) *v*, type(vector),intent(in),optional *v0*, real*4,intent(in),optional
eps, integer,intent(in),optional *iter_max*, logical,intent(in),optional *is_precond*)

solve $mx=v$ by jacobi

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
v : type(vector)
v0 : type(vector), initial vector (optional)
eps : real*4 , tolerance error (optional)
iter_max : integer, iteration max (optional)
is_precond : if .true -> with preconditionnor (optional)

Date

15th of March, 2011

Remarks**Returns**

res : type(t_soleq)

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 359 of file linear_equation.f90.

12.2.3.11 `type(vector) mod_linear_equation::leq_leastSquare (type(matrix),intent(in) m,
type(vector),intent(in) v)`

solve $mx=v$ by least square

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

Date

15th of March, 2011

Remarks**Returns**

res : type(vector)

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$

Definition at line 208 of file linear_equation.f90.

12.2.3.12 `type(vector) mod_linear_equation::leq_leastSquare_QR (type(matrix),intent(in) m, type(vector),intent(in) v, character*(*) ,intent(in),optional meth_qr, real*4,intent(in),optional eps_gsortho, logical,intent(in),optional is_permuted)`

solve $mx=v$ by least square

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

meth_qr : character*(*) ('ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization) (optional)

eps_gsortho : real*4 , precision g-s reortho... (optional)

is_permuted : with permutation matrix (optional)

Date

27th of March, 2011

Remarks

Returns

res : type(vector)

Exceptions

- + • m and v must be allocated
- *m*%rows = *v*%size\$

Definition at line 246 of file linear_equation.f90.

12.2.3.13 `type(vector) mod_linear_equation::leq_lu (type(matrix),intent(in) m, type(vector),intent(in) v, logical,intent(in),optional is_permuted)`

solve $mx=v$ by lu decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks**Returns**

res : type(vector)

See also

[math_machine_eps\(\)](#) : epsilon error

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 1574 of file linear_equation.f90.

12.2.3.14 `type(vector) mod_linear_equation::leq_precond_diag (type(matrix),intent(in) m,
type(vector),intent(in) v)`

preconditionnor : diagonal of m

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

Date

14th of March, 2011

Remarks

diag(m) must not null

Returns

res : vector

Definition at line 1782 of file linear_equation.f90.

12.2.3.15 `type(vector) mod_linear_equation::leq_precond_sor (type(matrix),intent(in) m,
type(vector),intent(in) v, type(vector),intent(in) v0, real*4,intent(in) w_relax)`

preconditionnor : diagonal of m

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
v : type(vector)
v0 : type(vector)
w_relax : relaxation w

Date

14th of March, 2011

Remarks

diag(m) must not null

Returns

res : vector

Definition at line 1804 of file linear_equation.f90.

12.2.3.16 `type(vector) mod_linear_equation::leq_pseudoinverse (type(matrix),intent(in) m,
type(vector),intent(in) v, real*4,intent(in),optional eps_svd, real*4,intent(in),optional
eps_chol, integer,intent(in),optional iter_max, character*(*),intent(in),optional
meth_qr, real*4,intent(in),optional eps_gsortho, character*(*),intent(in),optional
meth_pinv)`

solve $mx=v$ by least square

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
v : type(vector)
eps_svd : real*4 , precision svd (optional)
eps_chol : real*4 , optional tolerance error for cholseky factorization
iter_max : maximum iteration (optional)
meth_qr : character*(*), 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization (optional)
eps_gsortho : real*4 , precision gramm-schimdt reortho... (optional)
meth_pinv : character*(*), 'svd'=>svd decomposition or 'chol'=> cholesky decompositiion (optional)

Date

30th of March, 2011

Remarks

Returns

res : type(vector)

Exceptions

- + • m and v must be allocated
- $m\%rows = v\%size$

Definition at line 303 of file linear_equation.f90.

12.2.3.17 `type(vector) mod_linear_equation::leq_qr (type(matrix),intent(in) m,
type(vector),intent(in) v, character*(*),intent(in),optional meth_qr,
real*4,intent(in),optional eps_gsorto, logical,intent(in),optional is_permuted)`

solve $mx=v$ by qr decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

meth_qr : character*(*), 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization (optional)

eps_gsorto : real*4 , precision g-s reortho... (optional)

is_permuted : with permutation matrix (optional)

Date

26th of March, 2011

Remarks**Returns**

res : type(vector)

See also

[math_machine_eps\(\)](#) : epsilon error

Exceptions

- + • m and v must be allocated
- $m\%rows = v\%size$
- $m\%rows = m\%cols$ (square matrix expected)

Definition at line 1725 of file linear_equation.f90.

12.2.3.18 `type(vector) mod_linear_equation::leq_solve (type(matrix),intent(in) m,
type(vector),intent(in) v, character*(*),intent(in),optional meth_solve,
type(vector),intent(in),optional v0, real*4,intent(in),optional eps_svd,
real*4,intent(in),optional eps_chol, integer,intent(in),optional iter_max,
character*(*),intent(in),optional meth_qr, character*(*),intent(in),optional meth_pinv,
real*4,intent(in),optional eps_gsortho, logical,intent(in),optional is_permuted)`

solve linear equation

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
v : type(vector)
meth_solve : character*(*), solve method
v0 : type(vector), initial vector (optional)
eps_svd : real*4 , precision svd (optional)
eps_chol : real*4 , optional tolerance error for cholseky factorization
iter_max : maximum iteration (optional)
meth_qr : character*(*), 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization, (optional)
meth_pinv : character*(*), 'svd'=>svd decomposition or 'chol'=> cholesky decompositiion (optional)
eps_gsortho : real*4 , precision g-s reortho... (optional)
is_permuted : with permutation matrix (optional)

Date

01th of April, 2011

Remarks

classical method

Returns

res : type(vector), vector solution

Definition at line 1836 of file linear_equation.f90.

12.2.3.19 `type(t_soleq) mod_linear_equation::leq_solve_iter (type(matrix),intent(in) m,
type(vector),intent(in) v, character*(*),intent(in),optional meth_solve,
type(vector),intent(in),optional v0, real*4,intent(in),optional eps,
integer,intent(in),optional iter_max, logical,intent(in),optional is_precond,
real*4,intent(in),optional w_relax, integer,dimension(:),intent(in),optional tab_block_i,
integer,intent(in),optional size_block_i)`

solve linear equation

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
v : type(vector)
meth_solve : character*(*), iterative solve method
v0 : type(vector), initial vector (optional)
eps : real*4 , tolerance error (optional)
iter_max : maximum iteration (optional)
is_precond : if .true -> with preconditionnor (optional)
w_relax : real*, relaxation coef
tab_block_i : block index
size_block_i : size block index

Date

01th of April, 2011

Remarks

iterativs method

Returns

res : type(vector), vector solution

Definition at line 1905 of file linear_equation.f90.

12.2.3.20 `type(t_soleq) mod_linear_equation::leq_sor_elt (type(matrix),intent(in) m,
type(vector),intent(in) v, type(vector),intent(in),optional v0, real*4,intent(in),optional
eps, integer,intent(in),optional iter_max, real*4,intent(in),optional w_relax,
logical,intent(in),optional is_precond)`

solve $mx=v$ by sor

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
v : type(vector)
v0 : type(vector), initial vector (optional)
eps : real*4 , tolerance error (optional)
iter_max : integer, iteration max (optional)
w_relax : real*, relaxation coef (optional)
is_precond : if .true -> with preconditionnor (optional)

Date

15th of March, 2011

Remarks**Returns**

res : type(t_soleq)

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 488 of file linear_equation.f90.

12.2.3.21 type(vector) mod_linear_equation::leq_thomas (type(matrix),intent(in) m, type(vector),intent(in) v)

solve $mx=v$ if m is tridiagonal

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

Date

15th of March, 2011

Remarks

matrix must be tridiagonal

Returns

res : type(vector)

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 1393 of file linear_equation.f90.

12.2.3.22 `type(vector) mod_linear_equation::leq_tril (type(matrix),intent(in) m, type(vector),intent(in) v)`

solve $mx=v$ if m is lower triangular (forward substitution)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

Date

25th of March, 2011

Remarks

matrix must be lower triangular

Returns

res : type(vector)

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 1466 of file linear_equation.f90.

12.2.3.23 `type(vector) mod_linear_equation::leq_triu (type(matrix),intent(in) m, type(vector),intent(in) v)`

solve $mx=v$ if m is upper triangular (back substitution)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

Date

25th of March, 2011

Remarks

matrix must be lower triangular

Returns

res : type(vector)

Exceptions

- + • m and v must be allocated
 - $m\%rows = v\%size$
 - $m\%rows = m\%cols$ (square matrix expected)

Definition at line 1519 of file linear_equation.f90.

12.2.3.24 subroutine mod_linear_equation::m_destruct_soleq (type(t_soleq) soleq_)

destruct a soleq

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

soleq_ : type(t_soleq)

Date

26th of March, 2011

Remarks

Definition at line 89 of file linear_equation.f90.

12.3 mod_maths Module Reference

module [mod_maths](#)

Functions/Subroutines

- real *4 [math_machine_eps](#) ()
function math_machine_eps(filename) result(res)

Variables

- character(len=100) [math_what_exception](#)
exception signification of maths

12.3.1 Detailed Description



module [mod_maths](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Version

1.0

Date

27th of March, 2011

module for mathematics functions

Remarks

depend : [fml_constants.h](#)

12.3.2 Description

...

12.3.3 Function/Subroutine Documentation

12.3.3.1 `real*4 mod_maths::math_machine_eps ()`

function `math_machine_eps(filename)` result(`res`)

calculate the machine epsilon

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

05th of April, 2011

Remarks

$$1.0 + \frac{1}{2^p} = 1.0$$

Returns

machine_eps, epsilon machine

Definition at line 32 of file maths.f90.

12.3.4 Variable Documentation**12.3.4.1 character(len= 100) mod_maths::math_what_exception**

exception signification of maths

See also

[fml_constants.h](#)

Definition at line 23 of file maths.f90.

12.4 mod_matrix Module Reference

module [mod_matrix](#)

Data Types

- interface [assignment\(=\)](#)
brief interface assignment(=)
- interface [chol](#)
interface generic --> chol
- interface [destruct](#)
interface generic --> destruct matrix and derived type
- interface [det](#)
interface generic --> det
- interface [diag](#)
interface generic --> diag
- interface [get](#)
interface generic --> get

- interface [init](#)
interface generic --> init
- interface [inv](#)
interface generic --> inv
- interface [lu](#)
interface generic --> lu
- interface [m_lu](#)
interface generic --> m_lu
- type [matrix](#)
type matrix
- interface [max](#)
interface generic --> max
- interface [min](#)
interface generic --> min
- interface [norm](#)
interface generic --> norm
- interface [operator\(*\)](#)
interface operator()*
- interface [operator\(+\)](#)
interface operator(+)
- interface [operator\(-\)](#)
interface operator(-)
- interface [operator\(.cond.\)](#)
interface operator(.cond.)
- interface [operator\(.det.\)](#)
interface operator(.det.)
- interface [operator\(.inv.\)](#)
interface operator(.inv.)
- interface [operator\(.rank.\)](#)
interface operator(.rank.)
- interface [operator\(.tr.\)](#)
interface operator(.tr.)
- interface [operator\(/\)](#)
interface operator(/)

- interface `operator(==)`
interface operator(==)
- interface `pinv`
interface generic --> pinv
- interface `print`
interface generic --> print
- interface `qr`
interface generic --> qr
- interface `random`
interface generic --> random
- interface `rank`
interface generic --> rank
- interface `set`
interface generic --> set
- interface `spec`
interface generic --> spec
- interface `sum`
interface generic --> sum
- interface `svd`
interface generic --> svd
- type `t_eig`
type t_eig
- type `t_lu`
type t_lu
- type `t_m_and_p`
type t_m_and_p
- type `t_poweig`
type t_poweig
- type `t_qr`
type t_qr
- type `t_svd`
type t_svd
- interface `tril`

interface generic --> tril

- interface [triu](#)

interface generic --> triu

Functions/Subroutines

- type([matrix](#)) [m_add](#) (m1, m2)
- subroutine [m_affect](#) (m, value)
- type([matrix](#)) [m_bidiag_low](#) (m)
- type([matrix](#)) [m_bidiag_up](#) (m)
- real *4 [m_cond](#) (m)
- type([t_m_and_p](#)) [m_decompCholesky](#) (m, is_permuted)
- type([t_lu](#)) [m_decompLU](#) (m, is_permuted)
- type([t_m_and_p](#)) [m_decompLU_m](#) (m, is_permuted)
- type([t_qr](#)) [m_decompQR](#) (m, meth_qr, eps_gsortho, is_permuted)
- type([t_qr](#)) [m_decompQR_GramSchmidt](#) (m, is_permuted)
- type([t_qr](#)) [m_decompQR_GramSchmidt_Reortho](#) (m, eps, is_permuted)
- type([t_qr](#)) [m_decompQR_Householder](#) (m, is_permuted)
- type([t_svd](#)) [m_decompsvd](#) (m, eps, iter_max, meth_qr, eps_gsortho, is_permuted)
- type([vector](#)) [m_decompsvd_s](#) (m, eps, iter_max, meth_qr, eps_gsortho, is_permuted)
- subroutine [m_destruct](#) (m)
- subroutine [m_destruct_lu](#) (lu_)
- subroutine [m_destruct_m_and_p](#) (m_and_p_)
- subroutine [m_destruct_qr](#) (qr_)
- subroutine [m_destruct_t_eig](#) (t_eig_)
- subroutine [m_destruct_t_poweig](#) (t_poweig_)
- subroutine [m_destruct_t_svd](#) (t_svd_)
- real *4 [m_det](#) (m, meth_det, is_permuted)
- real *4 [m_det_chol](#) (m, is_permuted)
- real *4 [m_det_gaussj](#) (m)
- real *4 [m_det_lu](#) (m, is_permuted)
- real *4 [m_det_lu_all](#) (m, is_permuted)
- type([vector](#)) [m_diag](#) (m, i)
- type([matrix](#)) [m_div_scalar](#) (m, alpha)
- type([t_eig](#)) [m_eig_deflation](#) (m, v0, eps, iter_max)
- type([vector](#)) [m_eig_qr](#) (m, iter_max, meth_qr, eps_gsortho, is_permuted)
- type([matrix](#)) [m_extract](#) (m, r_lbound, r_ubound, c_lbound, c_ubound)
- real *4 [m_get](#) (m, i, j)
- real *4, dimension(m%rows, m%cols) [m_get_m](#) (m)
- real *4, dimension(m%rows) [m_getCol](#) (m, j)
- real *4, dimension(m%cols) [m_getRow](#) (m, i)
- integer [m_getSize](#) (m)
- integer [m_getSizeCols](#) (m)
- integer [m_getSizeRows](#) (m)
- type([matrix](#)) [m_identity](#) (n)
- subroutine [m_init](#) (m, rows_, cols_)
- subroutine [m_init_fromfile](#) (m, filename, unit)
- type([matrix](#)) [m_inverse_gaussj](#) (m)

- logical [m_isEqual](#) (m1, m2)
- logical [m_isEqual_scalar](#) (m, val)
- logical [m_isSymmetric](#) (m)
- type([vector](#)) [m_matrixTOvector](#) (m)
- real *4 [m_max](#) (m)
- real *4 [m_maxCol](#) (m, j)
- real *4 [m_maxRow](#) (m, i)
- real *4 [m_min](#) (m)
- real *4 [m_minCol](#) (m, j)
- subroutine [m_minit_value](#) (m, value)
- real *4 [m_minRow](#) (m, i)
- type([matrix](#)) [m_minus](#) (m1, m2)
- integer [m_nbnegative](#) (m)
- integer [m_nbnegativeCol](#) (m, j)
- integer [m_nbnegativeRow](#) (m, i)
- integer [m_nbpositive](#) (m)
- integer [m_nbpositiveCol](#) (m, j)
- integer [m_nbpositiveRow](#) (m, i)
- integer [m_nbzeros](#) (m)
- integer [m_nbzerosCol](#) (m, j)
- integer [m_nbzerosRow](#) (m, i)
- real *4 [m_norm](#) (m, type_norm)
- type([t_m_and_p](#)) [m_permut](#) (m, is_permuted)
- type([t_m_and_p](#)) [m_permut_col](#) (m, is_permuted)
- type([matrix](#)) [m_pinv](#) (m, eps_svd, eps_chol, iter_max, meth_qr, eps_gsorto, meth_pinv)
- type([t_poweig](#)) [m_pow_eig](#) (m, v0, eps, iter_max)
- subroutine [m_print](#) (m)
- subroutine [m_print_lu_tofile](#) (lu_decomp, filename, unit, status, position)
- subroutine [m_print_m_and_p_tofile](#) (m_and_p_decomp, filename, unit, status, position)
- subroutine [m_print_qr_tofile](#) (qr_decomp, filename, unit, status, position)
- subroutine [m_print_tofile](#) (m, filename, unit, status, position)
- type([matrix](#)) [m_prod_mat](#) (m1, m2)
- type([matrix](#)) [m_prod_scalar1](#) (alpha, m)
- type([matrix](#)) [m_prod_scalar2](#) (m, alpha)
- type([vector](#)) [m_prod_vec1](#) (m, v)
- type([vector](#)) [m_prod_vec2](#) (v, m)
- type([vector](#)) [m_prod_vec_c](#) (m, v)
- type([matrix](#)) [m_pseudoinv_chol](#) (m, eps_chol)
- type([matrix](#)) [m_pseudoinv_svd](#) (m, eps_svd, iter_max, meth_qr, eps_gsorto)
- integer [m_rank](#) (m, tol_rank, meth_rk, eps_svd, iter_max, meth_qr, eps_gsorto, is_permuted)
- integer [m_rank_gaussj](#) (m)
- integer [m_rank_svd](#) (m, tol_rank, eps_svd, iter_max, meth_qr, eps_gsorto, is_permuted)
- subroutine [m_resize](#) (m, rows_, cols_)
- subroutine [m_set](#) (m, i, j, value)
- subroutine [m_setsub](#) (m, r_lbound, r_ubound, c_lbound, c_ubound, m_sub)
- real *4 [m_sum](#) (m)
- real *4 [m_sumCol](#) (m, j)
- real *4 [m_sumRow](#) (m, i)
- real *4 [m_trace](#) (m)
- type([matrix](#)) [m_trans](#) (m)

- type([matrix](#)) [m_tril](#) (m, swap_diag)
- type([matrix](#)) [m_triu](#) (m, swap_diag)
- type([matrix](#)) [m_zeros](#) (rows_, cols_)
- type([matrix](#)) [mc_bidiag_low](#) (d, l, size_n)
- type([matrix](#)) [mc_bidiag_up](#) (d, u, size_n)
- type([matrix](#)) [mc_diag](#) (vect_data)
- subroutine [mc_diagDominant](#) (m, n, low, high)
- subroutine [mc_diagDominantSymmetric](#) (m, n, low, high)
- subroutine [mc_random](#) (m, low, high)
- type([matrix](#)) [mc_tridiag](#) (d, u, l, size_n)

Variables

- integer, parameter [frobenius](#) = -1
- character(len=100) [m_what_exception](#)
brief exception signification of matrix

12.4.1 Detailed Description



module [mod_matrix](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Version

1.0

Date

09th of March, 2011

module for matrix library

Remarks

depend : [mod_matrix](#), [mod_exception](#), [fml_constants.h](#)

12.4.2 Description

This module "mod_matrix" defines a matrix and provides some analysis features vector.

See also

[mod_vector](#)
[mod_exception](#)
[mod_utility](#)
[fml_constants.h](#)

12.4.3 Function/Subroutine Documentation

12.4.3.1 `type(matrix) mod_matrix::m_add (type(matrix),intent(in) m1, type(matrix),intent(in) m2)`

compute addition between the both m1 and m2 matrices

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m1 : type(matrix)
m2 : type(matrix)

Date

12th of March, 2011

Remarks

Returns

res : type(matrix)

Definition at line 1978 of file matrix.f90.

12.4.3.2 `subroutine mod_matrix::m_affect (type(matrix),intent(inout) m, real*4,intent(in) value)`

initialize matrix to value

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
value : value

Date

13th of March, 2011

Remarks

Definition at line 1184 of file matrix.f90.

12.4.3.3 type(matrix) mod_matrix::m_bidiag_low (type(matrix),intent(in) *m*)

lower bidiagonal matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 2427 of file matrix.f90.

12.4.3.4 type(matrix) mod_matrix::m_bidiag_up (type(matrix),intent(in) *m*)

upper bidiagonal matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 2454 of file matrix.f90.

12.4.3.5 real*4 mod_matrix::m_cond (type(matrix),intent(in) *m*)

condition number (≥ 0)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

26th of March, 2011

Remarks

$\text{norm}(\text{inv}.A) * \text{norm}(A)$

Returns

res : norm of vector

Todo

verify advantage between fortran basic function and loop implementation
do with singular value ($=s[\min(m,n)-1]$)

Definition at line 4225 of file matrix.f90.

**12.4.3.6 type(t_m_and_p) mod_matrix::m_decompCholesky (type(matrix),intent(in) *m*,
logical,intent(in),optional *is_permuted*)**

lu decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks

matrix must be positive definite

Returns

res : type(t_m_and_p), cholesky decomposition

See also

[math_machine_eps\(\)](#) : epsilon error

Todo

fill-permutation matrix ?

Definition at line 3447 of file matrix.f90.

12.4.3.7 `type(t_lu) mod_matrix::m_decompLU (type(matrix),intent(in) m,
logical,intent(in),optional is_permuted)`

lu decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix), mrows x mcols

is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks**Returns**

res : type(t_lu), lu decomposition (matrix L and U + permutation)

Definition at line 3344 of file matrix.f90.

12.4.3.8 `type(t_m_and_p) mod_matrix::m_decompLU_m (type(matrix),intent(in) m,
logical,intent(in),optional is_permuted)`

lu decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks**Returns**

res : type(t_m_and_p), lu decomposition (matrix + permutation)

Definition at line 3402 of file matrix.f90.

12.4.3.9 `type(t_qr) mod_matrix::m_decompQR (type(matrix),intent(in) m,
character*(*),intent(in),optional meth_qr, real*4,intent(in),optional eps_gsorto,
logical,intent(in),optional is_permuted)`

qr decomposition (HouseHolder or Schmidt)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

meth_qr : character*(*), 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization (optional)

eps_gsorto : real*4 , precision g-s reortho... (optional)

is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks

The Rayleigh Quotient Iteration is a natural union of the power and inverse power iterations

Returns

res : type(t_qr), QR decomposition

Todo

implement Givens method

Definition at line 3812 of file matrix.f90.

12.4.3.10 `type(t_qr) mod_matrix::m_decompQR_GramSchmidt (type(matrix),intent(in) m,
logical,intent(in),optional is_permuted)`

qr decomposition by Gram-Schmidt (bad in numerical computing)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
is_permuted : with permutation matrix (optional)

Date

26th of March, 2011

Remarks

The Rayleigh Quotient Iteration is a natural union of the power and inverse power iterations algorithm of Yousef Saad (eig_book_2ndEd)

```
A (m x n), if m>=n: * Q (m x n) orthornormal (.tr.Q.Q=I)
                  * R (n x n) upper triangular with nonnegative diagonal elements
```

Returns

res : type(t_qr), QR decomposition

See also

[math_machine_eps\(\)](#)

Todo

rank? Q and R sizes

Definition at line 3513 of file matrix.f90.

12.4.3.11 `type(t_qr) mod_matrix::m_decompQR_GramSchmidt_Reortho (`
`type(matrix),intent(in) m, real*4,intent(in),optional eps, logical,intent(in),optional`
`is_permuted)`

qr decomposition by Gram-Schmidt (reorthogonalization)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
is_permuted : with permutation matrix (optional)
eps : real*4 , tolerance error (optional)

Date

03th of April, 2011

Remarks

The Rayleigh Quotient Iteration is a natural union of the power and inverse power iterations algorithm of Yousef Saad (eig_book_2ndEd)

See also

[math_machine_eps\(\)](#)

```

computes the QR decomposition using modified Gram-Schmidt
with reorthogonalization
z is a vector that counts the reorthogonalization
steps per column (source Ritzit)

A (m x n), if m>=n:
    * Q (m x n) orthornormal (.tr.Q.Q=I)
    * R (n x n) upper triangular with nonnegative diagonal elements
Algorithms for the QR-Decomposition (Walter Gander)

```

delete or save z vector

Definition at line 3620 of file matrix.f90.

12.4.3.12 **type(t_qr) mod_matrix::m_decompQR_HouseHolder (type(matrix),intent(in) m, logical,intent(in),optional is_permuted)**

qr decomposition by HouseHolder (bad in numerical computing)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
is_permuted : with permutation matrix (optional)

Date

26th of March, 2011

Remarks

The Rayleigh Quotient Iteration is a natural union of the power and inverse power iterations algorithm of Yousef Saad (eig_book_2ndEd) A (m x n), * Q (m x m) orthornormal (.tr.Q.Q=I) * R (m x n) upper triangular with nonnegative diagonal elements

Returns

res : type(t_qr), QR decomposition

See also

[math_machine_eps\(\)](#)

Definition at line 3722 of file matrix.f90.

12.4.3.13 **type(t_svd) mod_matrix::m_decompsvd (type(matrix),intent(in) m, real*4,intent(in),optional eps, integer,intent(in),optional iter_max, character*(*),intent(in),optional meth_qr, real*4,intent(in),optional eps_gsortho, logical,intent(in),optional is_permuted)**

svd decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
eps : real*4 , precision (optional)
eps_gsortho : real*4 , precision g-s reortho... (optional)
iter_max : integer, iteration max (optional)
meth_qr : character*(*), 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization (optional)
is_permuted : with permutation matrix (optional)

Date

28th of March, 2011

Remarks

$A = U * S * V'$, $S \geq 0$, $U' * U = I_u$, and $V' * V = I_v$ * S : type(matrix), diagonal matrix (singular values) * V : type(matrix), orthogonal or unitary square matrix $V' * V = I_v$ * D : type(matrix), orthogonal or unitary square matrix $D' * D = Id$ Algorithm of '% Paul Godfrey % October 23, 2006 matlab (svdsim.m)' only work with HouseHolder QR decomposition or Gram-Schmidt with $m < n$

Returns

res : type(t_svd), svd decomposition

Definition at line 3853 of file matrix.f90.

12.4.3.14 type(vector) mod_matrix::m_decompsvd_s (type(matrix),intent(in)
m, real*4,intent(in),optional *eps*, integer,intent(in),optional *iter_max*,
character*(*),intent(in),optional *meth_qr*, real*4,intent(in),optional *eps_gsortho*,
logical,intent(in),optional *is_permuted*)

svd decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
eps : real*4 , precision (optional)
iter_max : integer, iteration max (optional)
meth_qr : character*(*), 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization (optional)
eps_gsortho : real*4 , precision g-s reortho... (optional)
is_permuted : with permutation matrix (optional)

Date

30th of March, 2011

Remarks**See also**[m_decompsvd](#)**Returns**

res : type(vector), singular values vector

Definition at line 3943 of file matrix.f90.

12.4.3.15 subroutine mod_matrix::m_destruct (type(matrix) *m*)

destruct a matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

09th of March, 2011

Remarks

Definition at line 710 of file matrix.f90.

12.4.3.16 subroutine mod_matrix::m_destruct_lu (type(t_lu) *lu_*)

destruct a lu

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

lu_ : type(t_lu)

Date

14th of March, 2011

Remarks

Definition at line 737 of file matrix.f90.

12.4.3.17 subroutine mod_matrix::m_destruct_m_and_p (type(t_m_and_p) *m_and_p_*)

destruct a lu

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m_and_p_ : type(t_m_and_p)

Date

25th of March, 2011

Remarks

Definition at line 751 of file matrix.f90.

12.4.3.18 subroutine mod_matrix::m_destruct_qr (type(t_qr) *qr_*)

destruct a qr

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

qr_ : type(t_qr)

Date

26th of March, 2011

Remarks

Definition at line 764 of file matrix.f90.

12.4.3.19 subroutine mod_matrix::m_destruct_t_eig (type(t_eig) *t_eig_*)

destruct a [t_eig](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

t_eig_ : type(t_eig)

Date

26th of March, 2011

Remarks

Definition at line 792 of file matrix.f90.

12.4.3.20 subroutine mod_matrix::m_destruct_t_poweig (type(t_poweig) t_poweig_)

destruct a [t_poweig](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

t_poweig_ : type(t_poweig)

Date

26th of March, 2011

Remarks

Definition at line 779 of file matrix.f90.

12.4.3.21 subroutine mod_matrix::m_destruct_t_svd (type(t_svd) t_svd_)

destruct a [t_svd](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

t_svd_ : type(t_svd)

Date

28th of March, 2011

Remarks

Definition at line 805 of file matrix.f90.

12.4.3.22 `real*4 mod_matrix::m_det (type(matrix),intent(in) m, character*(*),intent(in),optional meth_det, logical,intent(in),optional is_permuted)`

rk decomposition (HouseHolder or Schmidt)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

meth_det : character*(*), 'gj'=>gaussjourdan or 'chol'=>chol decomposition or 'lu'=>lu decomposition (optional)

is_permuted : with permutation matrix (optional)

Date

05th of April, 2011

Remarks

Returns

det : real*4 , the determinant

Definition at line 2558 of file matrix.f90.

12.4.3.23 `real*4 mod_matrix::m_det_chol (type(matrix),intent(in) m, logical,intent(in),optional is_permuted)`

compute the determinant by cholesky decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks

Returns

res : real*4 , the determinant

Todo

check the determinant computing

Definition at line 2653 of file matrix.f90.

12.4.3.24 real*4 mod_matrix::m_det_gaussj (type(matrix),intent(in) *m*)

determinant of *m* by gauss-jourdan

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

det : real*4

See also

[math_machine_eps\(\)](#) : epsilon error

Method

In order to compute matrix determinant, we can use matrices elementary determinant, following well-known:

- $L_i \leftrightarrow L_j, \det E_{ij} = -1$
- $L_i \rightarrow cL_i, \det E_i(c) = c, c \neq 0$
- $L_i \leftrightarrow L_j + cL_i, \det E_{ij} = 1, c \neq 0$

Definition at line 2695 of file matrix.f90.

12.4.3.25 real*4 mod_matrix::m_det_lu (type(matrix),intent(in) *m*, logical,intent(in),optional *is_permuted*)

compute the determinant by lu decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks**Returns**

res : real*4 , the determinant

Definition at line 2584 of file matrix.f90.

12.4.3.26 real*4 mod_matrix::m_det_lu_all (type(matrix),intent(in) m, logical,intent(in),optional is_permuted)

compute the determinant by lu decomposition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks

matrix can be rectangular

Returns

res : real*4 , the determinant

Definition at line 2620 of file matrix.f90.

12.4.3.27 type(vector) mod_matrix::m_diag (type(matrix),intent(in) m, integer,intent(in),optional i)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

i : i-th diagonal (-:low;+:top), optional

Date

12th of March, 2011

Remarks

Returns

res : type(vector), diagonal

Todo

optimize the if conditions

Definition at line 2485 of file matrix.f90.

12.4.3.28 type(matrix) mod_matrix::m_div_scalar (type(matrix),intent(in) *m*, real*4,intent(in) *alpha*)

calculate multiplication between the both m matrice and a scalar

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

alpha : scalar

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 1945 of file matrix.f90.

12.4.3.29 type(t_eig) mod_matrix::m_eig_deflation (type(matrix),intent(in) *m*, type(vector),intent(in),optional *v0*, real*4,intent(in),optional *eps*, integer,intent(in),optional *iter_max*)

matrix eigen values and vectors

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v0 : type(vector), initial vector (optional)

eps : real*4 , tolerance error (optional)

iter_max : integer, iteration max (optional)

Date

27th of March, 2011

Remarks

spectral radius or `m_t_eig`

Returns

`res` : `type(t_eig)`

See also

`type(t_eig)`

Todo

do not work fine (after the second eigen value)

Definition at line 4117 of file `matrix.f90`.

12.4.3.30 `type(vector) mod_matrix::m_eig_qr (type(matrix),intent(in) m, integer,intent(in) iter_max, character*(*),intent(in),optional meth_qr, real*4,intent(in),optional eps_gsortho, logical,intent(in),optional is_permuted)`

eigenvalue by QR decompositions

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : `type(matrix)`

iter_max

meth_qr : `character*(*)`, 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization (optional)

eps_gsortho : `real*4` , precision g-s reortho... (optional)

is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks**Returns**

`res` : `type(vector)`, `eig`

Definition at line 4171 of file `matrix.f90`.

12.4.3.31 `type(matrix) mod_matrix::m_extract (type(matrix),intent(in) m,
integer,intent(in),optional r_lbound, integer,intent(in),optional r_ubound,
integer,intent(in),optional c_lbound, integer,intent(in),optional c_ubound)`

extract sub matrix of

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
r_lbound : lower row boundary (optional)
r_ubound : upper row boundary (optional)
c_lbound : lower column boundary (optional)
c_ubound : upper column boundary (optional)

Date

29th of March, 2011

Remarks

Returns

res : sub matrix element

Exceptions

boundary :
 • $1 \geq r_lbound \leq m$

Definition at line 925 of file matrix.f90.

12.4.3.32 `real*4 mod_matrix::m_get (type(matrix),intent(in) m, integer,intent(in) i,
integer,intent(in) j)`

get the (i-th,j-th) matrix's element

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
i : i-th row
j : i-th col

Date

13th of March, 2011

Remarks

Returns

res : (i-th,j-th) matrix's element

Definition at line 1055 of file matrix.f90.

12.4.3.33 `real*4,dimension(m%rows,m%cols) mod_matrix::m_get_m (type(matrix),intent(in) m)`

get the matrix container

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

13th of March, 2011

Remarks**Returns**

res : matrix container

Definition at line 1081 of file matrix.f90.

12.4.3.34 `real*4,dimension(m%rows) mod_matrix::m_getCol (type(matrix),intent(in) m, integer,intent(in) j)`

get j-th col matrix's element

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

j : j-th col

Date

13th of March, 2011

Remarks**Returns**

res : j-th col matrix's element

Definition at line 1130 of file matrix.f90.

12.4.3.35 `real*4,dimension(m%cols) mod_matrix::m_getRow (type(matrix),intent(in) m,
integer,intent(in) i)`

get i-th row matrix's element

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

i : i-th row

Date

13th of March, 2011

Remarks**Returns**

res : i-th row matrix's element

Definition at line 1103 of file matrix.f90.

12.4.3.36 `integer mod_matrix::m_getSize (type(matrix),intent(in) m)`

return size of m

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

nb : size of matrix

Definition at line 1209 of file matrix.f90.

12.4.3.37 `integer mod_matrix::m_getSizeCols (type(matrix),intent(in) m)`

return size of a

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

nb : number of columns

Definition at line 1230 of file matrix.f90.

12.4.3.38 integer mod_matrix::m_getSizeRows (type(matrix),intent(in) m)

return size of a

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

nb : number of rows

Definition at line 1251 of file matrix.f90.

12.4.3.39 type(matrix) mod_matrix::m_identity (integer,intent(in) n)

create identity matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

n : size of identity matrix

Date

12th of March, 2011

Remarks

Returns

res : type(matrix)

Definition at line 2147 of file matrix.f90.

12.4.3.40 subroutine mod_matrix::m_init (type(matrix) m, integer,intent(in),optional rows_, integer,intent(in),optional cols_)

init a matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

rows_ : row (optional, default: =2)

cols_ : column (optional, default: =2)

Date

09th of March, 2011

Remarks**Exceptions**

positive size expected for rows_ and cols_

Definition at line 469 of file matrix.f90.

12.4.3.41 subroutine mod_matrix::m_init_fromfile (type(matrix) m, character*(*),intent(in) filename, integer,intent(in),optional unit)

init a matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

filename : name of file

unit : write unity (optional)

Date

12th of March, 2011

Remarks

Definition at line 516 of file matrix.f90.

12.4.3.42 `type(matrix) mod_matrix::m_inverse_gaussj (type(matrix),intent(in) m)`

inverse of *m* by gauss-jourdan

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

See also

[math_machine_eps\(\)](#) : epsilon error

Definition at line 2928 of file matrix.f90.

12.4.3.43 `logical mod_matrix::m_isEqual (type(matrix),intent(in) m1, type(matrix),intent(in) m2)`

verify if the matrix is Equal

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m1 : type(matrix)

m2 : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

res : logical

Definition at line 2070 of file matrix.f90.

12.4.3.44 logical mod_matrix::m_isEqual_scalar (type(matrix),intent(in) *m*, real*4,intent(in) *val*)

verify if the matrix is Equal

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

val : scalar

Date

12th of March, 2011

Remarks**Returns**

res : logical

Definition at line 2096 of file matrix.f90.

12.4.3.45 logical mod_matrix::m_isSymmetric (type(matrix),intent(in) *m*)

verify if the matrix is symmetric

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

res : logical

Definition at line 2119 of file matrix.f90.

12.4.3.46 type(vector) mod_matrix::m_matrixTOvector (type(matrix),intent(in) *m*)

transform matrix to vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

29th of March, 2011

Remarks**Returns**

res : vector element of m

Todo

add exception

Definition at line 1026 of file matrix.f90.

12.4.3.47 real*4 mod_matrix::m_max (type(matrix),intent(in) m)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks

res : type(matrix)

Returns

val : max of matrix

Definition at line 1499 of file matrix.f90.

12.4.3.48 real*4 mod_matrix::m_maxCol (type(matrix),intent(in) m , integer,intent(in) j)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

j : j-th col

Date

12th of March, 2011

Remarks**Returns**

val : max of matrix (j-th col)

Definition at line 1549 of file matrix.f90.

12.4.3.49 real*4 mod_matrix::m_maxRow (type(matrix),intent(in) m, integer,intent(in) i)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

i : i-th row

Date

12th of March, 2011

Remarks**Returns**

val : max of matrix (i-th row)

Definition at line 1521 of file matrix.f90.

12.4.3.50 real*4 mod_matrix::m_min (type(matrix),intent(in) m)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks

res : type(matrix)

Returns

val : min of matrix

Definition at line 1576 of file matrix.f90.

12.4.3.51 `real*4 mod_matrix::m_minCol (type(matrix),intent(in) m, integer,intent(in) j)`**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

j : j-th col

Date

12th of March, 2011

Remarks**Returns**

val : min of matrix (j-th col)

Definition at line 1627 of file matrix.f90.

12.4.3.52 `subroutine mod_matrix::m_init_value (type(matrix) m, real*4,intent(in) value)`

init matrix by scalar

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

value : scalar

Date

12th of March, 2011

Remarks

Definition at line 863 of file matrix.f90.

12.4.3.53 `real*4 mod_matrix::m_minRow (type(matrix),intent(in) m, integer,intent(in) i)`**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

i : i-th row

Date

12th of March, 2011

Remarks**Returns**

val : min of matrix (i-th row)

Definition at line 1598 of file matrix.f90.

**12.4.3.54 type(matrix) mod_matrix::m_minus (type(matrix),intent(in) m1,
type(matrix),intent(in) m2)**

compute soustraction between the both m1 and m2 matrices

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m1 : type(matrix)

m2 : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 2010 of file matrix.f90.

12.4.3.55 integer mod_matrix::m_nbnegative (type(matrix),intent(in) m)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

nb : occurence of negative's matrix value

Definition at line 1273 of file matrix.f90.

12.4.3.56 `integer mod_matrix::m_nbnegativeCol (type(matrix),intent(in) m, integer,intent(in) j)`

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

j : j-th col

Date

12th of March, 2011

Remarks

Returns

nb : occurence of negative's matrix j-th col value

Definition at line 1322 of file matrix.f90.

12.4.3.57 `integer mod_matrix::m_nbnegativeRow (type(matrix),intent(in) m, integer,intent(in) i)`

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

i : i-th row

Date

12th of March, 2011

Remarks

Returns

nb : occurence of negative's matrix i-th row value

Definition at line 1295 of file matrix.f90.

12.4.3.58 `integer mod_matrix::m_nbpositive (type(matrix),intent(in) m)`

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks

res : type(matrix)

Returns

nb : occurrence of positive's matrix value

Definition at line 1348 of file matrix.f90.

12.4.3.59 integer mod_matrix::m_nbpositiveCol (type(matrix),intent(in) m, integer,intent(in) j)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

j : j-th col

Date

12th of March, 2011

Remarks**Returns**

nb : occurrence of positive's matrix j-th col value

Definition at line 1398 of file matrix.f90.

12.4.3.60 integer mod_matrix::m_nbpositiveRow (type(matrix),intent(in) m, integer,intent(in) i)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

i : i-th row

Date

12th of March, 2011

Remarks**Returns**

nb : occurrence of positive's matrix i-th row value

Definition at line 1370 of file matrix.f90.

12.4.3.61 integer mod_matrix::m_nbzeros (type(matrix),intent(in) *m*)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

nb : occurence of zeros's matrix value

Definition at line 1424 of file matrix.f90.

12.4.3.62 integer mod_matrix::m_nbzerosCol (type(matrix),intent(in) *m*, integer,intent(in) *j*)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

j : j-th col

Date

12th of March, 2011

Remarks**Returns**

nb : the occurence of zeros's matrix value (j-th column)

Definition at line 1446 of file matrix.f90.

12.4.3.63 integer mod_matrix::m_nbzerosRow (type(matrix),intent(in) *m*, integer,intent(in) *i*)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

i : i-th Row

Date

12th of March, 2011

Remarks**Returns**

nb : the occurrence of zeros's matrix value (*j*-th row)

Definition at line 1473 of file matrix.f90.

12.4.3.64 **real*4 mod_matrix::m_norm (type(matrix),intent(in) *m*, integer,intent(in),optional *type_norm*)**

norm of matrix (≥ 0)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

type_norm : type of norm (optional)

Date

26th of March, 2011

Remarks

if *type_norm*=0, compute infy norm if *type_norm*=-1, compute frobenius norm

Returns

res : norm of matrix

Todo

verify advantage between fortran basic function and loop implementation

Definition at line 4254 of file matrix.f90.

12.4.3.65 **type(t_m_and_p) mod_matrix::m_permut (type(matrix),intent(in) *m*, logical,intent(in),optional *is_permuted*)**

permutation matrix (line)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks

line permutation

Returns

res : type(t_m_and_p)

Definition at line 3218 of file matrix.f90.

12.4.3.66 `type(t_m_and_p) mod_matrix::m_permut_col (type(matrix),intent(in) m,
logical,intent(in),optional is_permuted)`

permutation matrix (column)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
is_permuted : with permutation matrix (optional)

Date

25th of March, 2011

Remarks

column permutation

Returns

res : type(t_m_and_p)

Definition at line 3281 of file matrix.f90.

12.4.3.67 `type(matrix) mod_matrix::m_pinv (type(matrix),intent(in) m,
real*4,intent(in),optional eps_svd, real*4,intent(in),optional eps_chol,
integer,intent(in),optional iter_max, character*(*),intent(in),optional meth_qr,
real*4,intent(in),optional eps_gsortho, character*(*),intent(in),optional meth_pinv)`

pseudo inverse of m by svd

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
eps_svd : real*4 , precision svd (optional)
eps_chol : real*4 , optional tolerance error for cholseky factorization
iter_max : maximum iteration, for svd decomposition

meth_gr : character*(*), 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization, for svd decomposition (optional)
eps_gsortho : real*4, precision gramm-schimdt reortho..., for svd/qr decomposition (optional)
meth_pinv : character*(*), 'svd'=>svd decomposition or 'chol'=> cholesky decomposition (optional)

Date

05th of April, 2011

Remarks

pseudo inverse (by svd or cholesky decomposition method)

Returns

res : type(matrix) pseudo inverse of m

Étant donné une matrice A à coefficients réels ou complexes avec n lignes et p colonnes, son pseudo-inverse A + est l'unique matrice à p lignes et n colonnes vérifiant les conditions suivantes

1. $A A^+ A = A$;
2. $A^+ A A^+ = A^+$, (A + est un inverse pour le semi-groupe multiplicatif) ;
3. $(A A^+)^* = A A^+$, (A A + est une matrice hermitienne) ;
4. $(A^+ A)^* = A^+ A$, (A + A est également hermitienne) .

Definition at line 3026 of file matrix.f90.

12.4.3.68 **type(t_poweig) mod_matrix::m_pow_eig (type(matrix),intent(in) m, type(vector),intent(in),optional v0, real*4,intent(in),optional eps, integer,intent(in),optional iter_max)**

spectral radius (the maximum modulus of the eig)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
v0 : type(vector), initial vector (optional)
eps : real*4, precision (optional)
iter_max : integer, iteration max (optional)

Date

25th of March, 2011

Remarks

spectral radius or m_t_poweig

Returns

res : type(t_poweig)

See also

type(t_poweig)

Todo

treat complex case

Definition at line 4027 of file matrix.f90.

12.4.3.69 subroutine mod_matrix::m_print (type(matrix),intent(in) *m*)

display a matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

09th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 4308 of file matrix.f90.

**12.4.3.70 subroutine mod_matrix::m_print_lu_tofile (type(t_lu),intent(in) *lu_decomp*,
character*(*),intent(in) *filename*, integer,intent(in),optional *unit*,
character*(*),intent(in),optional *status*, character*(*),intent(in),optional *position*)**

print lu decomposition to file

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

lu_decomp : type(t_lu), lu decomposition

filename : name of file

unit : write unity (optional)

status : status (optional)

position : position of curseur (optional)

Date

27th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 4399 of file matrix.f90.

12.4.3.71 subroutine mod_matrix::m_print_m_and_p_tofile (type(t_m_and_p),intent(in)
m_and_p_decomp, character*(*),intent(in) *filename*, integer,intent(in),optional *unit*,
 character*(*),intent(in),optional *status*, character*(*),intent(in),optional *position*)

print lu decomposition to file

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m_and_p_decomp : type(t_m_and_p), lu decomposition
filename : name of file
unit : write unity (optional)
status : status (optional)
position : position of cursor (optional)

Date

27th of March, 2011

Remarks

Returns

res : type(matrix)

Definition at line 4523 of file matrix.f90.

12.4.3.72 subroutine mod_matrix::m_print_qr_tofile (type(t_qr),intent(in) *qr_decomp*,
 character*(*),intent(in) *filename*, integer,intent(in),optional *unit*,
 character*(*),intent(in),optional *status*, character*(*),intent(in),optional *position*)

print qr decomposition to file

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

qr_decomp : type(t_qr), lu decomposition
filename : name of file
unit : write unity (, optional)
status : status (optional)
position : position of curseur (optional)

Date

27th of March, 2011

Remarks

Returns

res : type(matrix)

Definition at line 4461 of file matrix.f90.

12.4.3.73 subroutine mod_matrix::m_print_toile (type(matrix),intent(in) *m*,
character*(*),intent(in) *filename*, integer,intent(in),optional *unit*,
character*(*),intent(in),optional *status*, character*(*),intent(in),optional *position*)

print matrix to file

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
filename : name of file
unit : write unity (optional)
status : status (optional)
position : position (optional)

Date

27th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 4339 of file matrix.f90.

12.4.3.74 type(matrix) mod_matrix::m_prod_mat (type(matrix),intent(in) *m1*,
type(matrix),intent(in) *m2*)

compute product between the both m1 and m2 matrices

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m1 : type(matrix)
m2 : type(matrix)

Date

09th of March, 2011

Remarks

Returns

res : type(matrix)

Definition at line 1740 of file matrix.f90.

**12.4.3.75 type(matrix) mod_matrix::m_prod_scalar1 (real*4,intent(in) *alpha*,
type(matrix),intent(in) *m*)**

compute multiplication between the both m matrice and a scalar

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

alpha : scalar

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 1895 of file matrix.f90.

**12.4.3.76 type(matrix) mod_matrix::m_prod_scalar2 (type(matrix),intent(in) *m*,
real*4,intent(in) *alpha*)**

compute multiplication between the both m matrice and a scalar

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

alpha : scalar

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 1920 of file matrix.f90.

12.4.3.77 `type(vector) mod_matrix::m_prod_vec1 (type(matrix),intent(in) m, type(vector),intent(in) v)`

compute product between matrix and vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

Date

13th of March, 2011

Remarks

Returns

res : type(vector)

Definition at line 1780 of file matrix.f90.

12.4.3.78 `type(vector) mod_matrix::m_prod_vec2 (type(vector),intent(in) v, type(matrix),intent(in) m)`

compute product between matrix and vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

m : type(matrix)

Date

13th of March, 2011

Remarks

Returns

res : type(vector)

Todo

parallel omp ? (ex, conjugate gradient)

Definition at line 1821 of file matrix.f90.

12.4.3.79 `type(vector) mod_matrix::m_prod_vec_c (type(matrix),intent(in) m,
type(vector),intent(in) v)`

compute product between matrix and vector (column compute)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

v : type(vector)

Date

13th of March, 2011

Remarks**Returns**

res : type(vector)

Todo

parallel omp ? (ex, conjugate gradient)

Definition at line 1861 of file matrix.f90.

12.4.3.80 `type(matrix) mod_matrix::m_pseudoinv_chol (type(matrix),intent(in) m,
real*4,intent(in),optional eps_chol)`

pseudo inverse of m by cholesky factorization

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

eps_chol : real*4 , optional tolerance error for cholseky factorization

Date

30th of March, 2011

Remarks

Moore-Penrose Inverse Matrices

Returns

res : type(matrix), Moore-Penrose inverse of m

Definition at line 3103 of file matrix.f90.

12.4.3.81 `type(matrix) mod_matrix::m_pseudoinv_svd (type(matrix),intent(in) m,
real*4,intent(in),optional eps_svd, integer,intent(in),optional iter_max,
character*(*),intent(in),optional meth_qr, real*4,intent(in),optional eps_gsortho)`

pseudo inverse of m by svd

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
eps_svd : real*4 , precision svd (optional)
iter_max : maximum iteration
meth_qr : character*(*), 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization (optional)
eps_gsortho : real*4 , precision g-s reortho... (optional)

Date

30th of March, 2011

Remarks

pseudo inverse by svd factorization : $A^+ = V * S^+ * tr(U)$

Returns

res : type(matrix) pseudo inverse of m

Definition at line 3060 of file matrix.f90.

12.4.3.82 `integer mod_matrix::m_rank (type(matrix),intent(in) m, real*4,intent(in),optional
tol_rank, character*(*),intent(in),optional meth_rk, real*4,intent(in),optional eps_svd,
integer,intent(in),optional iter_max, character*(*),intent(in),optional meth_qr,
real*4,intent(in),optional eps_gsortho, logical,intent(in),optional is_permuted)`

rk decomposition (HouseHolder or Schmidt)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
tol_rank : real*4 , precision rank (optional)
meth_rk : character*(*), 'gj'=>gaussjourdan or 'svd'=>svd decomposition (optional)
eps_svd : real*4 , precision svd (optional)
iter_max : maximum iteration
meth_qr : character*(*), 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization (optional)
eps_gsortho : real*4 , precision g-s reortho... (optional)
is_permuted : with permutation matrix (optional)

Date

30th of March, 2011

Remarks**Returns**

rank : real*4

Todo

implement Givens method

Definition at line 2895 of file matrix.f90.

12.4.3.83 integer mod_matrix::m_rank_gaussj (type(matrix),intent(in) m)

rank of m by gauss-jourdan

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

17th of March, 2011

Remarks

rank = number of non-zero value of the diagonal

Returns

rank : real*4

See also

[math_machine_eps\(\)](#) : epsilon error

Todo

it's not working well (use rank_svd)

Definition at line 2778 of file matrix.f90.

12.4.3.84 integer mod_matrix::m_rank_svd (type(matrix),intent(in) m, real*4,intent(in),optional tol_rank, real*4,intent(in),optional eps_svd, integer,intent(in),optional iter_max, character*(*),intent(in),optional meth_qr, real*4,intent(in),optional eps_gsortho, logical,intent(in),optional is_permuted)

rank of m by svd

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
tol_rank : real*4 , precision rank (optional)
eps_svd : real*4 , precision svd (optional)
iter_max : maximum iteration
meth_qr : character*(*), 'ho'=>householder or 'gs'=>gram-schmidt or 'gsro'=>gram-schmidt re-orthogonalization (optional)
eps_gsortho : real*4 , precision g-s reortho... (optional)
is_permuted : with permutation matrix (optional)

Date

29th of March, 2011

Remarks

rank (svd decomposition)

Returns

rank : real*4

Definition at line 2852 of file matrix.f90.

12.4.3.85 subroutine mod_matrix::m_resize (type(matrix),intent(inout) *m*, integer,intent(in) *rows_*, integer,intent(in) *cols_*)

resize matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
rows_ : row
cols_ : column

Date

14th of March, 2011

Remarks

Definition at line 671 of file matrix.f90.

12.4.3.86 subroutine mod_matrix::m_set (type(matrix),intent(inout) *m*, integer,intent(in) *i*, integer,intent(in) *j*, real*4,intent(in) *value*)

set the (i-th,j-th) matrix's element

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
i : i-th row
j : j-th col
value : value

Date

13th of March, 2011

Remarks

Definition at line 1158 of file matrix.f90.

12.4.3.87 subroutine mod_matrix::m_setsub (type(matrix),intent(inout) *m*,
integer,intent(in),optional *r_lbound*, integer,intent(in),optional *r_ubound*,
integer,intent(in),optional *c_lbound*, integer,intent(in),optional *c_ubound*,
type(matrix),intent(in) *m_sub*)

locally matrix updating

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
r_lbound : lower row boundary (optional)
r_ubound : upper row boundary (optional)
c_lbound : lower column boundary (optional)
c_ubound : upper column boundary (optional)
m_sub : submatrix

Date

29th of March, 2011

Remarks**Returns**

res : sub matrix element

Exceptions

boundary :

- $1 \geq r_lbound \leq m$

Definition at line 973 of file matrix.f90.

12.4.3.88 `real*4 mod_matrix::m_sum (type(matrix),intent(in) m)`**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

val : sum of matrix

Definition at line 1653 of file matrix.f90.

12.4.3.89 `real*4 mod_matrix::m_sumCol (type(matrix),intent(in) m, integer,intent(in) j)`**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
j : j-th col

Date

12th of March, 2011

Remarks**Returns**

val : sum of matrix (j-th col)

Definition at line 1702 of file matrix.f90.

12.4.3.90 `real*4 mod_matrix::m_sumRow (type(matrix),intent(in) m, integer,intent(in) i)`**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
i : i-th row

Date

12th of March, 2011

Remarks

Returns

val : sum of matrix i-th row

Definition at line 1675 of file matrix.f90.

12.4.3.91 real*4 mod_matrix::m_trace (type(matrix),intent(in) m)

compute the trace

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

res : real*4 , the matrix

Definition at line 3186 of file matrix.f90.

12.4.3.92 type(matrix) mod_matrix::m_trans (type(matrix),intent(in) m)

matrix transpose

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 2042 of file matrix.f90.

12.4.3.93 type(matrix) mod_matrix::m_tril (type(matrix),intent(in) m, integer,intent(in),optional swap_diag)

lower triangular matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
swap_diag : if swap_diag, without diagonal (optional)

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 2357 of file matrix.f90.

12.4.3.94 `type(matrix) mod_matrix::m_triu (type(matrix),intent(in) m,
integer,intent(in),optional swap_diag)`

upper triangular matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
swap_diag : if swap_diag, without diagonal (optional)

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 2393 of file matrix.f90.

12.4.3.95 `type(matrix) mod_matrix::m_zeros (integer,intent(in) rows_, integer,intent(in) cols_)`

create zeros matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

rows_ : row
cols_ : column

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 886 of file matrix.f90.

12.4.3.96 type(matrix) mod_matrix::mc_bidiag_low (real*4,intent(in) d, real*4,intent(in) l, integer,intent(in) size_n)

constructor of lower bidiagonal matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

d : diagonal value
l : lower diagonal value
size_n : size of the matrix (must be superior or equal 2)

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 2274 of file matrix.f90.

12.4.3.97 type(matrix) mod_matrix::mc_bidiag_up (real*4,intent(in) d, real*4,intent(in) u, integer,intent(in) size_n)

constructor of upper bidiagonal matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

d : diagonal value
u : upper diagonal value
size_n : size of the matrix (must be superior or equal 2)

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 2231 of file matrix.f90.

12.4.3.98 `type(matrix) mod_matrix::mc_diag (real*4,dimension(:) vect_data)`

constructor of diagonal matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

vect_data : a vector

Date

12th of March, 2011

Remarks**Returns**

res : type(matrix)

Definition at line 2315 of file matrix.f90.

12.4.3.99 `subroutine mod_matrix::mc_diagDominant (type(matrix) m, integer,intent(in) n, real*4,intent(in) low, real*4,intent(in) high)`

init a matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
n : matrix square size
low : lbound
high : ubound

Date

09th of March, 2011

Remarks

Definition at line 580 of file matrix.f90.

12.4.3.100 `subroutine mod_matrix::mc_diagDominantSymmetric (type(matrix) m, integer,intent(in) n, real*4,intent(in) low, real*4,intent(in) high)`

init a matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
n : matrix square size
low : lbound

high : ubound

Date

09th of March, 2011

Remarks

Definition at line 625 of file matrix.f90.

12.4.3.101 subroutine mod_matrix::mc_random (type(matrix),intent(inout) *m*, real*4,intent(in),optional *low*, real*4,intent(in),optional *high*)

initialize the matrix by random value

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

m : type(matrix)
low : lbound (optional, default=1.0)
high : ubound (optional, default=card(v))

Date

15th of March, 2011

Remarks

Definition at line 822 of file matrix.f90.

12.4.3.102 type(matrix) mod_matrix::mc_tridiag (real*4,intent(in) *d*, real*4,intent(in) *u*, real*4,intent(in) *l*, integer,intent(in) *size_n*)

constructor of tridiagonal matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

d : diagonal value
u : upper diagonal value
l : lower diagonal value
size_n : size of the matrix (must be superior or equal 2)

Date

12th of March, 2011

Remarks

Returns

res : type(matrix)

Definition at line 2186 of file matrix.f90.

12.4.4 Variable Documentation

12.4.4.1 integer,parameter mod_matrix::frobenius = -1

Definition at line 452 of file matrix.f90.

12.4.4.2 character(len= 100) mod_matrix::m_what_exception

brief exception signification of matrix

See also

[fml_constants.h](#)

Definition at line 449 of file matrix.f90.

12.5 mod_times Module Reference

module [mod_times](#)

Data Types

- type [t_time](#)
type *t_time*
- type [t_timeval](#)
type *t_timeval*

Functions/Subroutines

- subroutine [ft_begin](#) (timing_)
subroutine *ft_begin(timing_)*
- subroutine [ft_create_time](#) (timing_, name_, inbcalls_, ioffset_)
subroutine *ft_create_time(timing_name_,inbcalls_,ioffset_)*
- subroutine [ft_create_time_copy](#) (timing_, timingcopy_)
subroutine *ft_create_time_copy(timing_,timingcopy_)*
- subroutine [ft_create_time_init](#) (timing_)
subroutine *ft_create_time_init(timing_)*
- subroutine [ft_end](#) (timing_)
subroutine *ft_end(timing_)*
- subroutine [ft_gettimeofday](#) (t_timeval_)
subroutine *ft_gettimeofday(t_timeval_)*
- subroutine [ft_print](#) (timing_, myrank)
subroutine *ft_print(timing_ myrank)*
- subroutine [ft_reset](#) (timing_)
subroutine *ft_reset(timing_)*

12.5.1 Detailed Description



module [mod_times](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Version

1.0

Date

13th of September, 2010

module for times library

Remarks

depend : [fml_constants.h](#)

12.5.2 Description

...

12.5.3 Function/Subroutine Documentation

12.5.3.1 subroutine mod_times::ft_begin (type(t_time) *timing_*)

subroutine ft_begin(*timing_*)

start time

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

timing_ : type(t_time)

Date

20th of September, 2010

Remarks

Definition at line 204 of file times.f90.

12.5.3.2 subroutine mod_times::ft_create_time (type(t_time) *timing_*, character*(*) *name_*, integer *inbcalls_*, integer *ioffset_*)

subroutine ft_create_time(timing_,name_,inbcalls_,ioffset_)

constructor by values

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

timing_ : type(t_time)
name_ : name
inbcalls_ : number of call
ioffset_ : offset

Date

20th of September, 2010

Remarks

See also

timing.h

Definition at line 124 of file times.f90.

12.5.3.3 subroutine mod_times::ft_create_time_copy (type(t_time) *timing_*, type(t_time) *timingcopy_*)

subroutine ft_create_time_copy(timing_,timingcopy_)

constructor by copy

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

timing_ : type(t_time)
timingcopy_

Date

20th of September, 2010

Remarks

Definition at line 162 of file times.f90.

12.5.3.4 subroutine mod_times::ft_create_time_init (type(t_time) *timing_*)

subroutine ft_create_time_init(timing_)

init a type_time

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

timing_ : type(t_time)

Date

20th of September, 2010

Remarks

Definition at line 97 of file times.f90.

12.5.3.5 subroutine mod_times::ft_end (type(t_time) *timing_*)

subroutine ft_end(*timing_*)

end time

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

timing_ : type(t_time)

Date

20th of September, 2010

Remarks

Definition at line 225 of file times.f90.

12.5.3.6 subroutine mod_times::ft_gettimeofday (type(t_timeval) *t_timeval_*)

subroutine ft_gettimeofday(*t_timeval_*)

function shall obtain the current time, expressed as seconds and microseconds since the Epoch, and store it in the [t_timeval](#) structure pointed to by tp. The resolution of the system clock is unspecified.

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

t_timeval_ : type(t_timeval)

Date

20th of September, 2010

Remarks

Definition at line 78 of file times.f90.

12.5.3.7 subroutine mod_times::ft_print (type(t_time),intent(in) *timing_*, integer,intent(in),optional *myrank*)

subroutine ft_print(timing_, myrank)

display the time information

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

timing_ : type(t_time)
myrank : processor's rank (optional)

Date

20th of September, 2010

Remarks

Definition at line 250 of file times.f90.

12.5.3.8 subroutine mod_times::ft_reset (type(t_time) *timing_*)

subroutine ft_reset(timing_)

reset time

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

timing_ : type(t_time)

Date

20th of September, 2010

Remarks

Definition at line 188 of file times.f90.

12.6 mod_utility Module Reference

module [mod_utility](#)

Functions/Subroutines

- logical [u_is_exist_file](#) (filename)
function u_is_exist_file(filename) result(res)
- integer [u_nblne](#) (filename)
function u_nblne(filename) result(res)

Variables

- character(len=100) [u_what_exception](#)
exception signification of utility

12.6.1 Detailed Description



module [mod_utility](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Version

1.0

Date

27th of March, 2011

module for utilities functions

Remarks

depend : [fml_constants.h](#)

12.6.2 Description

...

12.6.3 Function/Subroutine Documentation

12.6.3.1 logical mod_utility::u_is_exist_file (character*(*),intent(in) *filename*)

function u_is_exist_file(filename) result(res)

test file existence

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

filename : name of file

Date

27th of March, 2011

Remarks**Returns**

boolean value (.true. if the file exist, .false. otherwise)

Definition at line 33 of file utility.f90.

12.6.3.2 integer mod_utility::u_nblne (character*(*),intent(in) *filename*)

function u_nblne(filename) result(res)

number line

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

filename : name of file

Date

27th of March, 2011

Remarks**Returns**

res : integer, number line

Definition at line 55 of file utility.f90.

12.6.4 Variable Documentation**12.6.4.1 character(len= 100) mod_utility::u_what_exception**

exception signification of utility

See also

[fml_constants.h](#)

Definition at line 23 of file utility.f90.

12.7 mod_vector Module Reference

module [mod_vector](#)

Data Types

- interface [abs](#)

- interface generic --> abs*
- interface [add](#)
interface generic --> add
- interface [assignment\(=\)](#)
brief interface assignment(=)
- interface [destruct](#)
interface generic --> destruct vector
- interface [dot](#)
interface generic --> dot
- interface [get](#)
interface generic --> get
- interface [init](#)
interface generic --> init
- interface [max](#)
interface generic --> max
- interface [min](#)
interface generic --> min
- interface [norm](#)
interface generic --> norm
- interface [operator\(*\)](#)
interface operator()*
- interface [operator\(+\)](#)
interface operator(+)
- interface [operator\(-\)](#)
interface operator(-)
- interface [operator\(.cross.\)](#)
interface operator(.cross.)
- interface [operator\(.dot.\)](#)
interface operator(.dot.)
- interface [operator\(.inv.\)](#)
interface operator(.inv.)
- interface [operator\(.len.\)](#)
interface operator(.len.)
- interface [operator\(.norm.\)](#)
interface operator(.norm.)
- interface [operator\(.sqnorm.\)](#)
interface operator(.sqnorm.)
- interface [operator\(/\)](#)
interface operator(/)
- interface [operator\(==\)](#)
interface operator(==)

- interface [print](#)
interface generic --> print
- interface [random](#)
interface generic --> random
- interface [set](#)
interface generic --> set
- interface [sqnorm](#)
interface generic --> sqnorm
- interface [sum](#)
interface generic --> sum
- type [vector](#)
*real*4 : type of precision : * real*4 for simple precision * real*8 for double precision*

Functions/Subroutines

- real *4 [v_abs](#) (v)
- type([vector](#)) [v_add](#) (v1, v2)
- subroutine [v_add_val_end](#) (v, val)
- subroutine [v_affect](#) (v, value)
- type([vector](#)) [v_axpby](#) (alpha, x, beta, y)
- type([vector](#)) [v_cross](#) (v1, v2)
- subroutine [v_destruct](#) (v)
- type([vector](#)) [v_div_scalar](#) (v, alpha)
- real *4 [v_dot](#) (v1, v2)
- type([vector](#)) [v_extract](#) (v, l_bound, u_bound)
- real *4 [v_get](#) (v, i)
- real *4, dimension(v%size) [v_get_v](#) (v)
- subroutine [v_init](#) (v, size_v)
- subroutine [v_init_fromfile](#) (v, filename, unit)
- subroutine [v_init_value](#) (v, value)
- type([vector](#)) [v_inverse](#) (v)
- logical [v_isEqual](#) (v1, v2)
- logical [v_isEqual_scalar](#) (v, val)
- real *4 [v_length](#) (v)
- real *4 [v_max](#) (v)
- real *4 [v_min](#) (v)
- type([vector](#)) [v_minus](#) (v1, v2)
- integer [v_nbnegative](#) (v)
- integer [v_nbpositive](#) (v)
- integer [v_nbzeros](#) (v)
- real *4 [v_norm](#) (v, type_norm)
- subroutine [v_normalize](#) (v)
- type([vector](#)) [v_ones](#) (size_v)
- subroutine [v_print](#) (v)
- subroutine [v_print_c](#) (v)
- subroutine [v_print_c_tofile](#) (v, filename, unit, status, position)
- subroutine [v_print_tofile](#) (v, filename, unit, status, position)
- real *4 [v_prod](#) (v)
- type([vector](#)) [v_prod_scalar1](#) (v, alpha)
- type([vector](#)) [v_prod_scalar2](#) (alpha, v)
- type([vector](#)) [v_prod_vec](#) (v1, v2)
- subroutine [v_resize](#) (v, size_v)
- subroutine [v_set](#) (v, i, value)
- integer [v_size](#) (v)
- real *4 [v_sqrtLength](#) (v)
- real *4 [v_sum](#) (v)
- type([vector](#)) [v_zeros](#) (size_v)
- subroutine [vc_random](#) (v, low, high)

Variables

- integer, parameter `infty = 0`
- character(len=100) `v_what_exception`

12.7.1 Detailed Description



module `mod_vector`

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Version

1.0

Date

09th of March, 2011

module for vector library

Remarks

depend : `mod_exception`, `fml_constants.h`

12.7.2 Description

This module "mod_vector" defines a vector and provides some analysis features vector.

See also

`mod_exception`
`mod_utility`
`fml_constants.h`

12.7.3 Function/Subroutine Documentation

12.7.3.1 `real*4 mod_vector::v_abs (type(vector),intent(in) v)`

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

`v` : type(vector)

Date

26th of March, 2011

Remarks**Returns**

res : vector absolute

Definition at line 865 of file vector.f90.

12.7.3.2 type(vector) mod_vector::v_add (type(vector),intent(in) v1, type(vector),intent(in) v2)

compute addition between the both m1 and m2 matrices

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v1 : type(vector)
v2 : type(vector)

Date

12th of March, 2011

Remarks**Returns**

res : type(vector)

Definition at line 1084 of file vector.f90.

12.7.3.3 subroutine mod_vector::v_add_val_end (type(vector),intent(inout) v, real*4,intent(in) val)

add val in the end of the vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
val : val to add into the vector v

Date

27th of March, 2011

Remarks

Definition at line 577 of file vector.f90.

12.7.3.4 subroutine mod_vector::v_affect (type(vector),intent(inout) *v*, real*4,intent(in) *value*)

initialize vector to value

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
value : value to add into the vector *v*

Date

13th of March, 2011

Remarks

Definition at line 714 of file vector.f90.

12.7.3.5 type(vector) mod_vector::v_axpby (real*4,intent(in) *alpha*, type(vector),intent(in) *x*, real*4,intent(in),optional *beta*, type(vector),intent(in),optional *y*)

compute product between the both *x* and *y* vectors

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

alpha : scalar
x : type(vector)
beta : scalar (optional)
y : type(vector) (optional)

Date

12th of March, 2011

Remarks**Returns**

res : type(vector)

Definition at line 1150 of file vector.f90.

12.7.3.6 type(vector) mod_vector::v_cross (type(vector),intent(in) *v1*, type(vector),intent(in) *v2*)

vector's cross

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v1 : type(vector)
v2 : type(vector)

Date

12th of March, 2011

Remarks**Returns**

res : vector's cross

Definition at line 1349 of file vector.f90.

12.7.3.7 subroutine mod_vector::v_destruct (type(vector) v)

destruct a vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks

Definition at line 426 of file vector.f90.

12.7.3.8 type(vector) mod_vector::v_div_scalar (type(vector),intent(in) v, real*4,intent(in) *alpha*)

ector divide by a scalar

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
alpha : real*4

Date

12th of March, 2011

Remarks

res

Returns

res : type(vector)

Definition at line 1050 of file vector.f90.

12.7.3.9 real*4 mod_vector::v_dot (type(vector),intent(in) v1, type(vector),intent(in) v2)

vector's dot

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v1 : type(vector)
v2 : type(vector)

Date

12th of March, 2011

Remarks**Returns**

res : vector's dot

Todo

openmp is better ?

Definition at line 1320 of file vector.f90.

12.7.3.10 type(vector) mod_vector::v_extract (type(vector),intent(in) v, integer,intent(in),optional l_bound, integer,intent(in),optional u_bound)

extract sub vector of

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
l_bound : lower boundary (optional)
u_bound : upper boundary (optional)

Date

29th of March, 2011

Remarks**Returns**

res : sub vector element

Exceptions

boundary :
 • $1 \geq l_bound \leq v$

Definition at line 618 of file vector.f90.

12.7.3.11 real*4 mod_vector::v_get (type(vector),intent(in) v, integer,intent(in) i)

get the i-th vector's element

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
 i : i-th element

Date

13th of March, 2011

Remarks**Returns**

res : i-th vector's element

Definition at line 640 of file vector.f90.

12.7.3.12 `real*4,dimension(v%size) mod_vector::v_get_v (type(vector),intent(in) v)`

get the vector container

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

13th of March, 2011

Remarks**Returns**

res : vector container

Definition at line 666 of file vector.f90.

12.7.3.13 `subroutine mod_vector::v_init (type(vector) v, integer,intent(in),optional size_v)`

init a vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

size_v : size vector (optional, default=1)

Date

12th of March, 2011

Remarks

Exceptions

positive size expected for *size_v*

Definition at line 305 of file vector.f90.

12.7.3.14 subroutine mod_vector::v_init_fromfile (type(vector) *v*, character*(*)*intent(in)* *filename*, integer*intent(in)*,optional *unit*)

init a vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
filename : name of file
unit : write unity (optional)

Date

12th of March, 2011

Remarks

Definition at line 343 of file vector.f90.

12.7.3.15 subroutine mod_vector::v_init_value (type(vector) *v*, real*4*intent(in)* *value*)

init vector by scalar

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
value : scalar

Date

12th of March, 2011

Remarks

res : type(vector)

Definition at line 528 of file vector.f90.

12.7.3.16 `type(vector) mod_vector::v_inverse (type(vector),intent(in) v)`

inverse of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

`v` : type(vector)

Date

14th of March, 2011

Remarks

`v` must not null

Returns

`res` : vector

Definition at line 940 of file vector.f90.

12.7.3.17 `logical mod_vector::v_isEqual (type(vector),intent(in) v1, type(vector),intent(in) v2)`

verify if the vector is Equal

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

`v1` : type(vector)
`v2` : type(vector)

Date

12th of March, 2011

Remarks**Returns**

`res` :logical

Definition at line 1378 of file vector.f90.

12.7.3.18 `logical mod_vector::v_isEqual_scalar (type(vector),intent(in) v, real*4,intent(in) val)`

verify if the vector is Equal

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
val : scalar

Date

12th of March, 2011

Remarks**Returns**

res :logical

Definition at line 1407 of file vector.f90.

12.7.3.19 real*4 mod_vector::v_length (type(vector),intent(in) v)

length of the vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks**Returns**

res : vector's length

Definition at line 1248 of file vector.f90.

12.7.3.20 real*4 mod_vector::v_max (type(vector),intent(in) v)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks**Returns**

val : max of vector

Definition at line 823 of file vector.f90.

12.7.3.21 real*4 mod_vector::v_min (type(vector),intent(in) v)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks**Returns**

val : min of vector

Definition at line 844 of file vector.f90.

12.7.3.22 type(vector) mod_vector::v_minus (type(vector),intent(in) v1, type(vector),intent(in) v2)

compute minusion between the both m1 and m2 matrices

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v1 : type(vector)
v2 : type(vector)

Date

12th of March, 2011

Remarks

Returns

res : type(vector)

Definition at line 1116 of file vector.f90.

12.7.3.23 integer mod_vector::v_nbnegative (type(vector),intent(in) v)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks**Returns**

nb : occurrence of negative's vector value

Definition at line 760 of file vector.f90.

12.7.3.24 integer mod_vector::v_nbpositive (type(vector),intent(in) v)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks**Returns**

nb : occurrence of positive's vector value

Definition at line 781 of file vector.f90.

12.7.3.25 integer mod_vector::v_nbzeros (type(vector),intent(in) v)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks**Returns**

nb : occurence of zeros's vector value

Definition at line 802 of file vector.f90.

12.7.3.26 real*4 mod_vector::v_norm (type(vector),intent(in) v, integer,intent(in),optional type_norm)

norm of vector (≥ 0)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
type_norm : type of norm (optional)

Date

26th of March, 2011

Remarks

if type_norm=0, compute infity norm

Returns

res : norm of vector

Definition at line 1277 of file vector.f90.

12.7.3.27 subroutine mod_vector::v_normalize (type(vector),intent(inout) v)

normalize the vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks

Definition at line 1199 of file vector.f90.

12.7.3.28 type(vector) mod_vector::v_ones (integer,intent(in) size_v)

create ones vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

size_v : vector size

Date

13th of March, 2011

Remarks**Returns**

res : type(vector)

Definition at line 495 of file vector.f90.

12.7.3.29 subroutine mod_vector::v_print (type(vector),intent(in) v)

display a vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

09th of March, 2011

Remarks

Returns

res : type(vector)

Definition at line 1440 of file vector.f90.

12.7.3.30 subroutine mod_vector::v_print_c (type(vector),intent(in) v)

display a vector (column view)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

14th of March, 2011

Remarks**Returns**

res : type(vector)

Definition at line 1520 of file vector.f90.

12.7.3.31 subroutine mod_vector::v_print_c_tofile (type(vector),intent(in) v, character*(*),intent(in) filename, integer,intent(in),optional unit, character*(*),intent(in),optional status, character*(*),intent(in),optional position)

print vector to file (column)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
filename : name of file
unit : write unity (optional)
status : status (optional)
position : position (optional)

Date

27th of March, 2011

Remarks**Returns**

res : type(vector)

Definition at line 1548 of file vector.f90.

12.7.3.32 subroutine `mod_vector::v_print_tofile` (`type(vector)`,`intent(in)` *v*,
`character*(*)`,`intent(in)` *filename*, `integer`,`intent(in)`,`optional` *unit*,
`character*(*)`,`intent(in)`,`optional` *status*, `character*(*)`,`intent(in)`,`optional` *position*)

print vector to file (row)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : `type(vector)`
filename : name of file
unit : write unity (optional)
status : status (optional)
position : position (optional)

Date

27th of March, 2011

Remarks

Returns

`res` : `type(vector)`

Definition at line 1465 of file `vector.f90`.

12.7.3.33 `real*4 mod_vector::v_prod` (`type(vector)`,`intent(in)` *v*)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : `type(vector)`

Date

13th of March, 2011

Remarks

Returns

`val` : prod of vector

Definition at line 911 of file `vector.f90`.

12.7.3.34 `type(vector) mod_vector::v_prod_scalar1 (type(vector),intent(in) v, real*4,intent(in) alpha)`

multiplication between vector and scalar

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
alpha : real*4

Date

12th of March, 2011

Remarks

res

Returns

res : type(vector)

Definition at line 1000 of file vector.f90.

12.7.3.35 `type(vector) mod_vector::v_prod_scalar2 (real*4,intent(in) alpha, type(vector),intent(in) v)`

multiplication between vector and scalar

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

alpha : real*4
v : type(vector)

Date

12th of March, 2011

Remarks**Returns**

res : type(vector)

Definition at line 1025 of file vector.f90.

12.7.3.36 `type(vector) mod_vector::v_prod_vec (type(vector),intent(in) v1,
type(vector),intent(in) v2)`

compute product between the both v1 and v2 vectors

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v1 : type(vector)
v2 : type(vector)

Date

12th of March, 2011

Remarks**Returns**

res : type(vector)

Definition at line 971 of file vector.f90.

12.7.3.37 `subroutine mod_vector::v_resize (type(vector),intent(inout) v, integer,intent(in) size_v
)`

resize vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
size_v : size vector

Date

14th of March, 2011

Remarks

Definition at line 393 of file vector.f90.

12.7.3.38 `subroutine mod_vector::v_set (type(vector),intent(inout) v, integer,intent(in) i,
real*4,intent(in) value)`

set the i-th element the vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
i : i-th element
value : element set into the vector

Date

13th of March, 2011

Remarks

Definition at line 688 of file vector.f90.

12.7.3.39 integer mod_vector::v_size (type(vector),intent(in) v)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks**Returns**

nb : size of vec

Definition at line 739 of file vector.f90.

12.7.3.40 real*4 mod_vector::v_sqrLength (type(vector),intent(in) v)

sqrLength the vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks**Returns**

res : vector's sqrLength

Definition at line 1219 of file vector.f90.

12.7.3.41 real*4 mod_vector::v_sum (type(vector),intent(in) v)**Author**

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)

Date

12th of March, 2011

Remarks**Returns**

val : sum of vector

Definition at line 889 of file vector.f90.

12.7.3.42 type(vector) mod_vector::v_zeros (integer,intent(in) size_v)

create zeros vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

size_v : vector size

Date

12th of March, 2011

Remarks**Returns**

res : type(vector)

Definition at line 550 of file vector.f90.

**12.7.3.43 subroutine mod_vector::vc_random (type(vector),intent(inout) v,
real*4,intent(in),optional low, real*4,intent(in),optional high)**

initialize the vector by random value

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v : type(vector)
low : lbound (optional, default=1.0)
high : ubound (optional, default=card(v))

Date

12th of March, 2011

Remarks

Definition at line 453 of file vector.f90.

12.7.4 Variable Documentation**12.7.4.1 integer,parameter mod_vector::infty = 0****See also**

[m_norm](#)

Definition at line 294 of file vector.f90.

12.7.4.2 character(len= 100) mod_vector::v_what_exception**See also**

[fml_constants.h](#)

Definition at line 290 of file vector.f90.

Chapter 13

Data Type Documentation

13.1 `mod_vector::abs` Interface Reference

interface generic --> abs

Public Member Functions

- real *4 [v_abs](#) (v)

13.1.1 Detailed Description

interface generic --> abs interfaces `v_abs` : absolute of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

26th of March, 2011

Remarks

Definition at line 212 of file `vector.f90`.

13.1.2 Member Function/Subroutine Documentation

13.1.2.1 `real*4 mod_vector::abs::v_abs (type(vector),intent(in) v)`

Definition at line 865 of file `vector.f90`.

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

- [vector.f90](#)

13.2 `mod_vector::add` Interface Reference

interface generic --> add

Public Member Functions

- subroutine [v_add_val_end](#) (v, val)

13.2.1 Detailed Description

interface generic --> add interfaces add : add value in vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 175 of file vector.f90.

13.2.2 Member Function/Subroutine Documentation

13.2.2.1 subroutine `mod_vector::add::v_add_val_end` (type(vector),intent(inout) v,
real*4,intent(in) val)

Definition at line 577 of file vector.f90.

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

- [vector.f90](#)

13.3 `mod_matrix::assignment(=)` Interface Reference

brief interface assignment(=)

Public Member Functions

- subroutine [m_affect](#) (m, value)

13.3.1 Detailed Description

brief interface assignment(=) interfaces = : affectation of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

13th of March, 2011

Remarks

Definition at line 185 of file matrix.f90.

13.3.2 Member Function/Subroutine Documentation

13.3.2.1 subroutine mod_matrix::assignment(=)::m_affect (type(matrix),intent(inout) *m*, real*4,intent(in) *value*)

Definition at line 1184 of file matrix.f90.

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

- [matrix.f90](#)

13.4 mod_vector::assignment(=) Interface Reference

brief interface assignment(=)

Public Member Functions

- subroutine [v_affect](#) (v, value)

13.4.1 Detailed Description

brief interface assignment(=) interfaces = : affectation of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

13th of March, 2011

Remarks

Definition at line 99 of file vector.f90.

13.4.2 Member Function/Subroutine Documentation

13.4.2.1 subroutine mod_vector::assignment(=)::v_affect (type(vector),intent(inout) v, real*4,intent(in) value)

Definition at line 714 of file vector.f90.

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

- [vector.f90](#)

13.5 mod_matrix::chol Interface Reference

interface generic --> chol

Public Member Functions

- [type\(t_m_and_p\) m_decompCholesky](#) (m, is_permuted)

13.5.1 Detailed Description

interface generic --> chol interfaces chol : cholesky factorisation

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 380 of file matrix.f90.

13.5.2 Member Function/Subroutine Documentation

13.5.2.1 type(t_m_and_p) mod_matrix::chol::m_decompCholesky (type(matrix),intent(in) m, logical,intent(in),optional is_permuted)

Definition at line 3447 of file matrix.f90.

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

- [matrix.f90](#)

13.6 mod_linear_equation::destruct Interface Reference

interface generic --> destruct linear_equation and derived type

Public Member Functions

- subroutine [m_destruct_soleq](#) (soleq_)

13.6.1 Detailed Description

interface generic --> destruct linear_equation and derived type interfaces destruct : destructor

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

26th of March, 2011

Remarks

Definition at line 73 of file linear_equation.f90.

13.6.2 Member Function/Subroutine Documentation

13.6.2.1 subroutine mod_linear_equation::destruct::m_destruct_soleq (type(t_soleq) soleq_)

Definition at line 89 of file linear_equation.f90.

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

- [linear_equation.f90](#)

13.7 mod_matrix::destruct Interface Reference

interface generic --> destruct matrix and derived type

Public Member Functions

- subroutine [m_destruct](#) (m)
- subroutine [m_destruct_lu](#) (lu_)
- subroutine [m_destruct_m_and_p](#) (m_and_p_)
- subroutine [m_destruct_qr](#) (qr_)
- subroutine [m_destruct_t_eig](#) (t_eig_)
- subroutine [m_destruct_t_poweig](#) (t_poweig_)
- subroutine [m_destruct_t_svd](#) (t_svd_)

13.7.1 Detailed Description

interface generic --> destruct matrix and derived type interfaces destruct : destructor

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

25th of March, 2011

Remarks

Definition at line 241 of file matrix.f90.

13.7.2 Member Function/Subroutine Documentation

13.7.2.1 subroutine mod_matrix::destruct::m_destruct (type(matrix) *m*)

Definition at line 710 of file matrix.f90.

13.7.2.2 subroutine mod_matrix::destruct::m_destruct_lu (type(t_lu) *lu_*)

Definition at line 737 of file matrix.f90.

13.7.2.3 subroutine mod_matrix::destruct::m_destruct_m_and_p (type(t_m_and_p) *m_and_p_*)

Definition at line 751 of file matrix.f90.

13.7.2.4 subroutine mod_matrix::destruct::m_destruct_qr (type(t_qr) *qr_*)

Definition at line 764 of file matrix.f90.

13.7.2.5 subroutine mod_matrix::destruct::m_destruct_t_eig (type(t_eig) *t_eig_*)

Definition at line 792 of file matrix.f90.

13.7.2.6 subroutine mod_matrix::destruct::m_destruct_t_poweig (type(t_poweig) *t_poweig_*)

Definition at line 779 of file matrix.f90.

13.7.2.7 subroutine mod_matrix::destruct::m_destruct_t_svd (type(t_svd) *t_svd_*)

Definition at line 805 of file matrix.f90.

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

- [matrix.f90](#)

13.8 `mod_vector::destruct` Interface Reference

interface generic --> destruct vector

Public Member Functions

- subroutine [v_destruct](#) (v)

13.8.1 Detailed Description

interface generic --> destruct vector interfaces destruct : destructor

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

25th of March, 2011

Remarks

Definition at line 166 of file `vector.f90`.

13.8.2 Member Function/Subroutine Documentation

13.8.2.1 subroutine `mod_vector::destruct::v_destruct` (type(vector) v)

Definition at line 426 of file `vector.f90`.

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

- [vector.f90](#)

13.9 `mod_matrix::det` Interface Reference

interface generic --> det

Public Member Functions

- real *4 [m_det](#) (m, meth_det, is_permuted)

13.9.1 Detailed Description

interface generic --> det interfaces det : det of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

30th of March, 2011

Remarks

Definition at line 371 of file matrix.f90.

13.9.2 Member Function/Subroutine Documentation

13.9.2.1 `real*4 mod_matrix::det::m_det (type(matrix),intent(in) m,
character*(*),intent(in),optional meth_det, logical,intent(in),optional is_permuted)`

Definition at line 2558 of file matrix.f90.

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

- [matrix.f90](#)

13.10 mod_matrix::diag Interface Reference

interface generic --> diag

Public Member Functions

- `type(vector) m_diag (m, i)`

13.10.1 Detailed Description

interface generic --> diag interfaces diag : diag of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

14th of March, 2011

Remarks

Definition at line 326 of file matrix.f90.

13.10.2 Member Function/Subroutine Documentation

13.10.2.1 `type(vector) mod_matrix::diag::m_diag (type(matrix),intent(in) m,
integer,intent(in),optional i)`

Definition at line 2485 of file matrix.f90.

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

- [matrix.f90](#)

13.11 mod_vector::dot Interface Reference

interface generic --> dot

Public Member Functions

- real *4 [v_dot](#) (v1, v2)

13.11.1 Detailed Description

interface generic --> dot interfaces dot : dot v1.v2 of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 221 of file vector.f90.

13.11.2 Member Function/Subroutine Documentation

13.11.2.1 real*4 mod_vector::dot::v_dot (type(vector),intent(in) v1, type(vector),intent(in) v2)

Definition at line 1320 of file vector.f90.

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

- [vector.f90](#)

13.12 mod_matrix::get Interface Reference

interface generic --> get

Public Member Functions

- real *4 [m_get](#) (m, i, j)
- real *4, dimension(m%rows, m%cols) [m_get_m](#) (m)
- real *4, dimension(m%cols) [m_getRow](#) (m, i)

13.12.1 Detailed Description

interface generic --> get interfaces get : get element of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 335 of file matrix.f90.

13.12.2 Member Function/Subroutine Documentation

13.12.2.1 `real*4 mod_matrix::get::m_get (type(matrix),intent(in) m, integer,intent(in) i, integer,intent(in) j)`

Definition at line 1055 of file matrix.f90.

13.12.2.2 `real*4,dimension(m%rows,m%cols) mod_matrix::get::m_get_m (type(matrix),intent(in) m)`

Definition at line 1081 of file matrix.f90.

13.12.2.3 `real*4,dimension(m%cols) mod_matrix::get::m_getRow (type(matrix),intent(in) m, integer,intent(in) i)`

Definition at line 1103 of file matrix.f90.

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

- [matrix.f90](#)

13.13 mod_vector::get Interface Reference

interface generic --> get

Public Member Functions

- `real *4 v_get (v, i)`
- `real *4, dimension(v%size) v_get_v (v)`

13.13.1 Detailed Description

interface generic --> get interfaces get : get element of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 248 of file vector.f90.

13.13.2 Member Function/Subroutine Documentation

13.13.2.1 `real*4 mod_vector::get::v_get (type(vector),intent(in) v, integer,intent(in) i)`

Definition at line 640 of file vector.f90.

13.13.2.2 `real*4,dimension(v%size) mod_vector::get::v_get_v (type(vector),intent(in) v)`

Definition at line 666 of file vector.f90.

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

- [vector.f90](#)

13.14 mod_linear_equation::ilinsolve Interface Reference

interface generic --> ilinsolve

Public Member Functions

- `type(t_oleq) leq_solve_iter` (m, v, meth_solve, v0, eps, iter_max, is_precond, w_relax, tab_block_i, size_block_i)

13.14.1 Detailed Description

interface generic --> ilinsolve interfaces ilinsolve : iteratives solve linear equation of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 55 of file linear_equation.f90.

13.14.2 Member Function/Subroutine Documentation

13.14.2.1 `type(t_soleq) mod_linear_equation::ilinsolve::leq_solve_iter (type(matrix),intent(in) m, type(vector),intent(in) v, character*(*),intent(in),optional meth_solve, type(vector),intent(in),optional v0, real*4,intent(in),optional eps, integer,intent(in),optional iter_max, logical,intent(in),optional is_precond, real*4,intent(in),optional w_relax, integer,dimension(:),intent(in),optional tab_block_i, integer,intent(in),optional size_block_i)`

Definition at line 1905 of file linear_equation.f90.

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

- [linear_equation.f90](#)

13.15 mod_vector::init Interface Reference

interface generic --> init

Public Member Functions

- subroutine [v_init](#) (v, size_v)
- subroutine [v_init_fromfile](#) (v, filename, unit)

13.15.1 Detailed Description

interface generic --> init interfaces init : init vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 275 of file vector.f90.

13.15.2 Member Function/Subroutine Documentation

13.15.2.1 `subroutine mod_vector::init::v_init (type(vector) v, integer,intent(in),optional size_v)`

Definition at line 305 of file vector.f90.

13.15.2.2 subroutine mod_vector::init::v_init_fromfile (type(vector) *v*, character*(*)*,intent(in) filename*, integer,intent(in),optional *unit*)

Definition at line 343 of file vector.f90.

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

- [vector.f90](#)

13.16 mod_matrix::init Interface Reference

interface generic --> init

Public Member Functions

- subroutine [m_init](#) (*m*, *rows_*, *cols_*)
- subroutine [m_init_fromfile](#) (*m*, *filename*, *unit*)

13.16.1 Detailed Description

interface generic --> init interfaces init : init vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 425 of file matrix.f90.

13.16.2 Member Function/Subroutine Documentation

13.16.2.1 subroutine mod_matrix::init::m_init (type(matrix) *m*, integer,intent(in),optional *rows_*, integer,intent(in),optional *cols_*)

Definition at line 469 of file matrix.f90.

13.16.2.2 subroutine mod_matrix::init::m_init_fromfile (type(matrix) *m*, character*(*)*,intent(in) filename*, integer,intent(in),optional *unit*)

Definition at line 516 of file matrix.f90.

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

- [matrix.f90](#)

13.17 mod_matrix::inv Interface Reference

interface generic --> inv

Public Member Functions

- type([matrix](#)) [m_inverse_gaussj](#) (m)

13.17.1 Detailed Description

interface generic --> inv interfaces inv: inverse of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

30th of March, 2011

Remarks

Definition at line 280 of file [matrix.f90](#).

13.17.2 Member Function/Subroutine Documentation

13.17.2.1 type([matrix](#)) mod_matrix::inv::m_inverse_gaussj (type([matrix](#)),intent(in) m)

Definition at line 2928 of file [matrix.f90](#).

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

- [matrix.f90](#)

13.18 mod_linear_equation::leq_getBlock Interface Reference

interface generic --> [leq_getBlock](#)

Public Member Functions

- type([matrix](#)) [gsb_getBlock_m](#) (m, r_s, r_e, c_s, c_e)
- type([vector](#)) [leq_getBlock_v](#) (v, r_s, r_e)

13.18.1 Detailed Description

interface generic --> [leq_getBlock](#) interfaces [leq_getBlock](#) : get block

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

14th of March, 2011

Remarks

Definition at line 64 of file linear_equation.f90.

13.18.2 Member Function/Subroutine Documentation

13.18.2.1 `type(matrix) mod_linear_equation::leq_getBlock::gsb_getBlock_m (`
`type(matrix),intent(in) m, integer,intent(in) r_s, integer,intent(in) r_e,`
`integer,intent(in) c_s, integer,intent(in) c_e)`

Definition at line 788 of file linear_equation.f90.

13.18.2.2 `type(vector) mod_linear_equation::leq_getBlock::leq_getBlock_v (`
`type(vector),intent(in) v, integer,intent(in) r_s, integer,intent(in) r_e)`

Definition at line 839 of file linear_equation.f90.

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

- [linear_equation.f90](#)

13.19 mod_linear_equation::linsolve Interface Reference

interface generic --> linsolve

Public Member Functions

- `type(vector) leq_solve (m, v, meth_solve, v0, eps_svd, eps_chol, iter_max, meth_qr, meth_pinv, eps_gsortho, is_permuted)`

13.19.1 Detailed Description

interface generic --> linsolve interfaces linsolve : solve linear equation of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks**Todo**

entraîné de les faire

Definition at line 46 of file linear_equation.f90.

13.19.2 Member Function/Subroutine Documentation

13.19.2.1 `type(vector) mod_linear_equation::linsolve::leq_solve (type(matrix),intent(in) m, type(vector),intent(in) v, character*(*),intent(in),optional meth_solve, type(vector),intent(in),optional v0, real*4,intent(in),optional eps_svd, real*4,intent(in),optional eps_chol, integer,intent(in),optional iter_max, character*(*),intent(in),optional meth_qr, character*(*),intent(in),optional meth_pinv, real*4,intent(in),optional eps_gsortho, logical,intent(in),optional is_permuted)`

Definition at line 1836 of file linear_equation.f90.

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

- [linear_equation.f90](#)

13.20 mod_matrix::lu Interface Reference

interface generic --> lu

Public Member Functions

- `type(t_lu) m_decompLU (m, is_permuted)`

13.20.1 Detailed Description

interface generic --> lu interfaces : lu factorisation

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 389 of file matrix.f90.

13.20.2 Member Function/Subroutine Documentation

13.20.2.1 `type(t_lu) mod_matrix::lu::m_decompLU (type(matrix),intent(in) m, logical,intent(in),optional is_permuted)`

Definition at line 3344 of file matrix.f90.

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

- [matrix.f90](#)

13.21 mod_matrix::m_lu Interface Reference

interface generic --> [m_lu](#)

Public Member Functions

- `type(t_m_and_p) m_decompLU_m (m, is_permuted)`

13.21.1 Detailed Description

interface generic --> [m_lu](#) interfaces : lu factorisation

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 398 of file matrix.f90.

13.21.2 Member Function/Subroutine Documentation

13.21.2.1 `type(t_m_and_p) mod_matrix::m_lu::m_decompLU_m (type(matrix),intent(in) m, logical,intent(in),optional is_permuted)`

Definition at line 3402 of file matrix.f90.

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

- [matrix.f90](#)

13.22 mod_matrix::matrix Type Reference

type matrix

Public Attributes

- integer `cols` = 2;
- real *4, dimension(:, :), allocatable `container`
- logical `is_allocate` = .false.
- real *4, dimension(:, :), allocatable `matrix`
- real *4, dimension(:, :), allocatable `ptr_container`
- integer `rows` = 2;

13.22.1 Detailed Description

type matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

rows : integer (by default =2), matrix row
cols : integer (by default =2), matrix col
ptr_container : matrix container
is_allocate : verify if the matrix is allocate

Date

09th of March, 2011

Remarks

Definition at line 41 of file matrix.f90.

13.22.2 Member Data Documentation

13.22.2.1 integer mod_matrix::matrix::cols = 2;

Definition at line 43 of file matrix.f90.

13.22.2.2 real*4,dimension(:, :),allocatable mod_matrix::matrix::container

Definition at line 44 of file matrix.f90.

13.22.2.3 logical mod_matrix::matrix::is_allocate = .false.

Definition at line 45 of file matrix.f90.

13.22.2.4 real*4,dimension(:, :),allocatable mod_matrix::matrix::matrix

Definition at line 44 of file matrix.f90.

13.22.2.5 real*4,dimension(:,:),allocatable mod_matrix::matrix::ptr_container

Definition at line 44 of file matrix.f90.

13.22.2.6 integer mod_matrix::matrix::rows = 2;

Definition at line 42 of file matrix.f90.

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

- [matrix.f90](#)

13.23 mod_vector::max Interface Reference

interface generic --> max

Public Member Functions

- real *4 [v_max](#) (v)

13.23.1 Detailed Description

interface generic --> max interfaces max : v of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 203 of file vector.f90.

13.23.2 Member Function/Subroutine Documentation**13.23.2.1 real*4 mod_vector::max::v_max (type(vector),intent(in) v)**

Definition at line 823 of file vector.f90.

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

- [vector.f90](#)

13.24 mod_matrix::max Interface Reference

interface generic --> max

Public Member Functions

- real *4 [m_max](#) (m)

13.24.1 Detailed Description

interface generic --> max interfaces max : v of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 271 of file matrix.f90.

13.24.2 Member Function/Subroutine Documentation

13.24.2.1 real*4 mod_matrix::max::m_max (type(matrix),intent(in) m)

Definition at line 1499 of file matrix.f90.

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

- [matrix.f90](#)

13.25 mod_vector::min Interface Reference

interface generic --> min

Public Member Functions

- real *4 [v_min](#) (v)

13.25.1 Detailed Description

interface generic --> min interfaces min : v of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 194 of file vector.f90.

13.25.2 Member Function/Subroutine Documentation

13.25.2.1 real*4 mod_vector::min::v_min (type(vector),intent(in) *v*)

Definition at line 844 of file vector.f90.

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

- [vector.f90](#)

13.26 mod_matrix::min Interface Reference

interface generic --> min

Public Member Functions

- real *4 [m_min](#) (*m*)

13.26.1 Detailed Description

interface generic --> min interfaces min : *v* of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 262 of file matrix.f90.

13.26.2 Member Function/Subroutine Documentation

13.26.2.1 real*4 mod_matrix::min::m_min (type(matrix),intent(in) *m*)

Definition at line 1576 of file matrix.f90.

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

- [matrix.f90](#)

13.27 mod_vector::norm Interface Reference

interface generic --> norm

Public Member Functions

- real *4 [v_norm](#) (v, type_norm)

13.27.1 Detailed Description

interface generic --> norm interfaces v_norm : norm p of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

26th of March, 2011

Remarks

Definition at line 230 of file vector.f90.

13.27.2 Member Function/Subroutine Documentation

13.27.2.1 `real*4 mod_vector::norm::v_norm (type(vector),intent(in) v, integer,intent(in),optional type_norm)`

Definition at line 1277 of file vector.f90.

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

- [vector.f90](#)

13.28 mod_matrix::norm Interface Reference

interface generic --> norm

Public Member Functions

- real *4 [m_norm](#) (m, type_norm)

13.28.1 Detailed Description

interface generic --> norm interfaces norm : norm p,q of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

26th of March, 2011

Remarks

Definition at line 299 of file `matrix.f90`.

13.28.2 Member Function/Subroutine Documentation

13.28.2.1 `real*4 mod_matrix::norm::m_norm (type(matrix),intent(in) m, integer,intent(in),optional type_norm)`

Definition at line 4254 of file `matrix.f90`.

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

- [matrix.f90](#)

13.29 `mod_vector::operator(*)` Interface Reference

interface `operator(*)`

Public Member Functions

- `type(vector) v_prod_scalar1 (v, alpha)`
- `type(vector) v_prod_scalar2 (alpha, v)`
- `type(vector) v_prod_vec (v1, v2)`

13.29.1 Detailed Description

interface `operator(*)` interfaces `prod_mat_mat` :

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

09th of March, 2011

Remarks

Definition at line 53 of file `vector.f90`.

13.29.2 Member Function/Subroutine Documentation

13.29.2.1 `type(vector) mod_vector::operator(*)::v_prod_scalar1 (type(vector),intent(in) v,
real*4,intent(in) alpha)`

Definition at line 1000 of file vector.f90.

13.29.2.2 `type(vector) mod_vector::operator(*)::v_prod_scalar2 (real*4,intent(in) alpha,
type(vector),intent(in) v)`

Definition at line 1025 of file vector.f90.

13.29.2.3 `type(vector) mod_vector::operator(*)::v_prod_vec (type(vector),intent(in) v1,
type(vector),intent(in) v2)`

Definition at line 971 of file vector.f90.

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

- [vector.f90](#)

13.30 mod_matrix::operator(*) Interface Reference

interface operator(*)

Public Member Functions

- `type(matrix) m_prod_mat (m1, m2)`
- `type(matrix) m_prod_scalar1 (alpha, m)`
- `type(matrix) m_prod_scalar2 (m, alpha)`
- `type(vector) m_prod_vec2 (v, m)`
- `type(vector) m_prod_vec_c (m, v)`

13.30.1 Detailed Description

interface operator(*) interfaces prod_mat_mat : product

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

09th of March, 2011

Remarks

Definition at line 140 of file matrix.f90.

13.30.2 Member Function/Subroutine Documentation

13.30.2.1 `type(matrix) mod_matrix::operator(*)::m_prod_mat (type(matrix),intent(in) m1,
type(matrix),intent(in) m2)`

Definition at line 1740 of file matrix.f90.

13.30.2.2 `type(matrix) mod_matrix::operator(*)::m_prod_scalar1 (real*4,intent(in) alpha,
type(matrix),intent(in) m)`

Definition at line 1895 of file matrix.f90.

13.30.2.3 `type(matrix) mod_matrix::operator(*)::m_prod_scalar2 (type(matrix),intent(in) m,
real*4,intent(in) alpha)`

Definition at line 1920 of file matrix.f90.

13.30.2.4 `type(vector) mod_matrix::operator(*)::m_prod_vec2 (type(vector),intent(in) v,
type(matrix),intent(in) m)`

Definition at line 1821 of file matrix.f90.

13.30.2.5 `type(vector) mod_matrix::operator(*)::m_prod_vec_c (type(matrix),intent(in) m,
type(vector),intent(in) v)`

Definition at line 1861 of file matrix.f90.

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

- [matrix.f90](#)

13.31 mod_matrix::operator(+) Interface Reference

interface operator(+)

Public Member Functions

- `type(matrix) m_add (m1, m2)`

13.31.1 Detailed Description

interface operator(+) interfaces add : addition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 167 of file matrix.f90.

13.31.2 Member Function/Subroutine Documentation**13.31.2.1** `type(matrix) mod_matrix::operator(+>::m_add (type(matrix),intent(in) m1,
type(matrix),intent(in) m2)`

Definition at line 1978 of file matrix.f90.

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

- [matrix.f90](#)

13.32 mod_vector::operator(+) Interface Reference

interface operator(+)

Public Member Functions

- `type(vector) v_add (v1, v2)`

13.32.1 Detailed Description

interface operator(+) interfaces add : addition

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 81 of file vector.f90.

13.32.2 Member Function/Subroutine Documentation**13.32.2.1** `type(vector) mod_vector::operator(+>::v_add (type(vector),intent(in) v1,
type(vector),intent(in) v2)`

Definition at line 1084 of file vector.f90.

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

- [vector.f90](#)

13.33 mod_matrix::operator(-) Interface Reference

interface operator(-)

Public Member Functions

- type([matrix](#)) [m_minus](#) (m1, m2)

13.33.1 Detailed Description

interface operator(-) interfaces minus : soustraction

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 176 of file matrix.f90.

13.33.2 Member Function/Subroutine Documentation

13.33.2.1 type([matrix](#)) mod_matrix::operator(-)::m_minus (type([matrix](#)),intent(in) *m1*,
type([matrix](#)),intent(in) *m2*)

Definition at line 2010 of file matrix.f90.

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

- [matrix.f90](#)

13.34 mod_vector::operator(-) Interface Reference

interface operator(-)

Public Member Functions

- type([vector](#)) [v_minus](#) (v1, v2)

13.34.1 Detailed Description

interface operator(-) interfaces minus : soustraction

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 90 of file vector.f90.

13.34.2 Member Function/Subroutine Documentation

13.34.2.1 `type(vector) mod_vector::operator(-)::v_minus (type(vector),intent(in) v1, type(vector),intent(in) v2)`

Definition at line 1116 of file vector.f90.

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

- [vector.f90](#)

13.35 `mod_matrix::operator(.cond.)` Interface Reference

`interface operator(.cond.)`

Public Member Functions

- `real *4 m_cond (m)`

13.35.1 Detailed Description

`interface operator(.cond.)` interfaces `cond` : `cond` of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

26th of March, 2011

Remarks

Definition at line 232 of file matrix.f90.

13.35.2 Member Function/Subroutine Documentation

13.35.2.1 real*4 mod_matrix::operator(.cond.):m_cond (type(matrix),intent(in) m)

Definition at line 4225 of file matrix.f90.

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

- [matrix.f90](#)

13.36 mod_vector::operator(.cross.) Interface Reference

```
interface operator(.cross.)
```

Public Member Functions

- type(vector) v_cross (v1, v2)

13.36.1 Detailed Description

interface operator(.cross.) interfaces .cross. : cross of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 138 of file vector.f90.

13.36.2 Member Function/Subroutine Documentation

13.36.2.1 type(vector) mod_vector::operator(.cross.):v_cross (type(vector),intent(in) v1, type(vector),intent(in) v2)

Definition at line 1349 of file vector.f90.

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

- [vector.f90](#)

13.37 mod_matrix::operator(.det.) Interface Reference

```
interface operator(.det.)
```

Public Member Functions

- real *4 [m_det_gaussj](#) (m)

13.37.1 Detailed Description

interface operator(.det.) interfaces det : determinant of matrix with gauss-jourdan method

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

17th of March, 2011

Remarks

Definition at line 204 of file matrix.f90.

13.37.2 Member Function/Subroutine Documentation

13.37.2.1 real*4 mod_matrix::operator(.det.):m_det_gaussj (type(matrix),intent(in) m)

Definition at line 2695 of file matrix.f90.

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

- [matrix.f90](#)

13.38 mod_vector::operator(.dot.) Interface Reference

interface operator(.dot.)

Public Member Functions

- real *4 [v_dot](#) (v1, v2)

13.38.1 Detailed Description

interface operator(.dot.) interfaces .dot. : dot of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 148 of file vector.f90.

13.38.2 Member Function/Subroutine Documentation**13.38.2.1** `real*4 mod_vector::operator(.dot.):v_dot (type(vector),intent(in) v1,
type(vector),intent(in) v2)`

Definition at line 1320 of file vector.f90.

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

- [vector.f90](#)

13.39 mod_matrix::operator(.inv.) Interface Reference

interface operator(.inv.)

Public Member Functions

- `type(matrix) m_inverse_gaussj (m)`

13.39.1 Detailed Description

interface operator(.inv.) interfaces inv : inverse of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 223 of file matrix.f90.

13.39.2 Member Function/Subroutine Documentation**13.39.2.1** `type(matrix) mod_matrix::operator(.inv.):m_inverse_gaussj (type(matrix),intent(in)
m)`

Definition at line 2928 of file matrix.f90.

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

- [matrix.f90](#)

13.40 `mod_vector::operator(.inv.)` Interface Reference

interface `operator(.inv.)`

Public Member Functions

- `type(vector) v_inverse (v)`

13.40.1 Detailed Description

interface `operator(.inv.)` interfaces `inv` : inverse of vector(if all element is not null)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

14th of March, 2011

Remarks

Definition at line 157 of file `vector.f90`.

13.40.2 Member Function/Subroutine Documentation

13.40.2.1 `type(vector) mod_vector::operator(.inv.)::v_inverse (type(vector),intent(in) v)`

Definition at line 940 of file `vector.f90`.

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

- [vector.f90](#)

13.41 `mod_vector::operator(.len.)` Interface Reference

interface `operator(.len.)`

Public Member Functions

- integer `v_size (v)`

13.41.1 Detailed Description

interface `operator(.len.)` interfaces `.len.` : length of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 109 of file `vector.f90`.

13.41.2 Member Function/Subroutine Documentation**13.41.2.1 `integer mod_vector::operator(.len.):v_size (type(vector),intent(in) v)`**

Definition at line 739 of file `vector.f90`.

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

- [vector.f90](#)

13.42 `mod_vector::operator(.norm.)` Interface Reference

`interface operator(.norm.)`

Public Member Functions

- `real *4 v_length (v)`

13.42.1 Detailed Description

`interface operator(.norm.)` interfaces `.norm.` : norm 2 of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 118 of file `vector.f90`.

13.42.2 Member Function/Subroutine Documentation**13.42.2.1 `real*4 mod_vector::operator(.norm.):v_length (type(vector),intent(in) v)`**

Definition at line 1248 of file `vector.f90`.

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

- [vector.f90](#)

13.43 mod_matrix::operator(.rank.) Interface Reference

interface operator(.rank.)

Public Member Functions

- integer [m_rank_gaussj](#) (m)

13.43.1 Detailed Description

interface operator(.rank.) interfaces .rank. : rank of matrix with gauss-jourdan method

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

17th of March, 2011

Remarks

Definition at line 214 of file matrix.f90.

13.43.2 Member Function/Subroutine Documentation

13.43.2.1 integer mod_matrix::operator(.rank.)::m_rank_gaussj (type(matrix),intent(in) m)

Definition at line 2778 of file matrix.f90.

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

- [matrix.f90](#)

13.44 mod_vector::operator(.sqnorm.) Interface Reference

interface operator(.sqnorm.)

Public Member Functions

- real *4 [v_sqrLength](#) (v)

13.44.1 Detailed Description

interface operator(.sqnorm.) interfaces .sqnorm. : square norm 2 of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 127 of file `vector.f90`.

13.44.2 Member Function/Subroutine Documentation**13.44.2.1 `real*4 mod_vector::operator(.sqnorm.)::v_sqrLength (type(vector),intent(in) v)`**

Definition at line 1219 of file `vector.f90`.

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

- [vector.f90](#)

13.45 `mod_matrix::operator(.tr.)` Interface Reference

`interface operator(.tr.)`

Public Member Functions

- `type(matrix) m_trans (m)`

13.45.1 Detailed Description

`interface operator(.tr.)` interfaces `trans` : transpose of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 195 of file `matrix.f90`.

13.45.2 Member Function/Subroutine Documentation**13.45.2.1 `type(matrix) mod_matrix::operator(.tr.)::m_trans (type(matrix),intent(in) m)`**

Definition at line 2042 of file `matrix.f90`.

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

- [matrix.f90](#)

13.46 mod_matrix::operator(/) Interface Reference

interface operator(/)

Public Member Functions

- type([matrix](#)) [m_div_scalar](#) (m, alpha)

13.46.1 Detailed Description

interface operator(/) interfaces prod_mat_mat : product

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

09th of March, 2011

Remarks

Definition at line 149 of file matrix.f90.

13.46.2 Member Function/Subroutine Documentation

13.46.2.1 type(matrix) mod_matrix::operator(/)::m_div_scalar (type(matrix),intent(in) m, real*4,intent(in) *alpha*)

Definition at line 1945 of file matrix.f90.

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

- [matrix.f90](#)

13.47 mod_vector::operator(/) Interface Reference

interface operator(/)

Public Member Functions

- type([vector](#)) [v_div_scalar](#) (v, alpha)

13.47.1 Detailed Description

interface operator(/) interfaces div_mat_scalar : divide by sclar

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 62 of file vector.f90.

13.47.2 Member Function/Subroutine Documentation

13.47.2.1 `type(vector) mod_vector::operator(/)::v_div_scalar (type(vector),intent(in) v, real*4,intent(in) alpha)`

Definition at line 1050 of file vector.f90.

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

- [vector.f90](#)

13.48 mod_matrix::operator(==) Interface Reference

interface operator(==)

Public Member Functions

- logical [m_isEqual](#) (m1, m2)
- logical [m_isEqual_scalar](#) (m, val)

13.48.1 Detailed Description

interface operator(==) interfaces isEqual : equal matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 158 of file matrix.f90.

13.48.2 Member Function/Subroutine Documentation

13.48.2.1 logical `mod_matrix::operator(==)::m_isEqual` (`type(matrix),intent(in) m1`, `type(matrix),intent(in) m2`)

Definition at line 2070 of file `matrix.f90`.

13.48.2.2 logical `mod_matrix::operator(==)::m_isEqual_scalar` (`type(matrix),intent(in) m`, `real*4,intent(in) val`)

Definition at line 2096 of file `matrix.f90`.

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

- [matrix.f90](#)

13.49 `mod_vector::operator(==)` Interface Reference

interface `operator(==)`

Public Member Functions

- logical [v_isEqual](#) (`v1`, `v2`)
- logical [v_isEqual_scalar](#) (`v`, `val`)

13.49.1 Detailed Description

interface `operator(==)` interfaces `isEqual` : equal vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 72 of file `vector.f90`.

13.49.2 Member Function/Subroutine Documentation

13.49.2.1 logical `mod_vector::operator(==)::v_isEqual` (`type(vector),intent(in) v1`, `type(vector),intent(in) v2`)

Definition at line 1378 of file `vector.f90`.

13.49.2.2 logical mod_vector::operator(==)::v_isEqual_scalar (type(vector),intent(in) v, real*4,intent(in) val)

Definition at line 1407 of file vector.f90.

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

- [vector.f90](#)

13.50 mod_matrix::pinv Interface Reference

interface generic --> pinv

Public Member Functions

- type(matrix) [m_pinv](#) (m, eps_svd, eps_chol, iter_max, meth_qr, eps_gsortho, meth_pinv)

13.50.1 Detailed Description

interface generic --> pinv interfaces pinv: pseudo inverse of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

30th of March, 2011

Remarks

Definition at line 289 of file matrix.f90.

13.50.2 Member Function/Subroutine Documentation

13.50.2.1 type(matrix) mod_matrix::pinv::m_pinv (type(matrix),intent(in) m, real*4,intent(in),optional eps_svd, real*4,intent(in),optional eps_chol, integer,intent(in),optional iter_max, character*(*),intent(in),optional meth_qr, real*4,intent(in),optional eps_gsortho, character*(*),intent(in),optional meth_pinv)

Definition at line 3026 of file matrix.f90.

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

- [matrix.f90](#)

13.51 mod_matrix::print Interface Reference

interface generic --> print

Public Member Functions

- subroutine [m_print](#) (m)
- subroutine [m_print_lu_tofile](#) (lu_decomp, filename, unit, status, position)
- subroutine [m_print_tofile](#) (m, filename, unit, status, position)

13.51.1 Detailed Description

interface generic --> print interfaces print : display matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

13th of March, 2011

Remarks

Definition at line 443 of file matrix.f90.

13.51.2 Member Function/Subroutine Documentation

13.51.2.1 subroutine mod_matrix::print::m_print (type(matrix),intent(in) m)

Definition at line 4308 of file matrix.f90.

13.51.2.2 subroutine mod_matrix::print::m_print_lu_tofile (type(t_lu),intent(in) lu_decomp, character*(*),intent(in) filename, integer,intent(in),optional unit, character*(*),intent(in),optional status, character*(*),intent(in),optional position)

Definition at line 4399 of file matrix.f90.

13.51.2.3 subroutine mod_matrix::print::m_print_tofile (type(matrix),intent(in) m, character*(*),intent(in) filename, integer,intent(in),optional unit, character*(*),intent(in),optional status, character*(*),intent(in),optional position)

Definition at line 4339 of file matrix.f90.

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

- [matrix.f90](#)

13.52 mod_vector::print Interface Reference

interface generic --> print

Public Member Functions

- subroutine [v_print](#) (v)
- subroutine [v_print_tofile](#) (v, filename, unit, status, position)

13.52.1 Detailed Description

interface generic --> print interfaces print : display matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

13th of March, 2011

Remarks

Definition at line 284 of file vector.f90.

13.52.2 Member Function/Subroutine Documentation

13.52.2.1 subroutine mod_vector::print::v_print (type(vector),intent(in) v)

Definition at line 1440 of file vector.f90.

13.52.2.2 subroutine mod_vector::print::v_print_tofile (type(vector),intent(in) v, character*(*),intent(in) filename, integer,intent(in),optional unit, character*(*),intent(in),optional status, character*(*),intent(in),optional position)

Definition at line 1465 of file vector.f90.

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

- [vector.f90](#)

13.53 mod_matrix::qr Interface Reference

interface generic --> qr

Public Member Functions

- type([t_qr](#)) [m_decompQR](#) (m, meth_qr, eps_gsortho, is_permuted)

13.53.1 Detailed Description

interface generic --> qr interfaces : qr factorisation

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 407 of file matrix.f90.

13.53.2 Member Function/Subroutine Documentation

13.53.2.1 `type(t_qr) mod_matrix::qr::m_decompQR (type(matrix),intent(in) m,
character*(*),intent(in),optional meth_qr, real*4,intent(in),optional eps_gsortho,
logical,intent(in),optional is_permuted)`

Definition at line 3812 of file matrix.f90.

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

- [matrix.f90](#)

13.54 mod_vector::random Interface Reference

interface generic --> random

Public Member Functions

- subroutine [vc_random](#) (v, low, high)

13.54.1 Detailed Description

interface generic --> random interfaces random : initialize by random values

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 266 of file vector.f90.

13.54.2 Member Function/Subroutine Documentation

13.54.2.1 subroutine mod_vector::random::vc_random (type(vector),intent(inout) v, real*4,intent(in),optional low, real*4,intent(in),optional high)

Definition at line 453 of file vector.f90.

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

- [vector.f90](#)

13.55 mod_matrix::random Interface Reference

interface generic --> random

Public Member Functions

- subroutine [mc_random](#) (m, low, high)

13.55.1 Detailed Description

interface generic --> random interfaces random : initialize by random values

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 434 of file matrix.f90.

13.55.2 Member Function/Subroutine Documentation

13.55.2.1 subroutine mod_matrix::random::mc_random (type(matrix),intent(inout) m, real*4,intent(in),optional low, real*4,intent(in),optional high)

Definition at line 822 of file matrix.f90.

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

- [matrix.f90](#)

13.56 mod_matrix::rank Interface Reference

interface generic --> rank

Public Member Functions

- integer [m_rank](#) (m, tol_rank, meth_rk, eps_svd, iter_max, meth_qr, eps_gsortho, is_permuted)

13.56.1 Detailed Description

interface generic --> rank interfaces rank : rank of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

30th of March, 2011

Remarks

Definition at line 362 of file matrix.f90.

13.56.2 Member Function/Subroutine Documentation

13.56.2.1 integer `mod_matrix::rank::m_rank` (type(matrix),intent(in) *m*,
real*4,intent(in),optional *tol_rank*, character*(*),intent(in),optional *meth_rk*,
real*4,intent(in),optional *eps_svd*, integer,intent(in),optional *iter_max*,
character*(*),intent(in),optional *meth_qr*, real*4,intent(in),optional *eps_gsortho*,
logical,intent(in),optional *is_permuted*)

Definition at line 2895 of file matrix.f90.

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

- [matrix.f90](#)

13.57 mod_vector::set Interface Reference

interface generic --> set

Public Member Functions

- subroutine [v_set](#) (v, i, value)

13.57.1 Detailed Description

interface generic --> set interfaces v_set : set element of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 257 of file vector.f90.

13.57.2 Member Function/Subroutine Documentation

13.57.2.1 `subroutine mod_vector::set::v_set (type(vector),intent(inout) v, integer,intent(in) i, real*4,intent(in) value)`

Definition at line 688 of file vector.f90.

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

- [vector.f90](#)

13.58 mod_matrix::set Interface Reference

interface generic --> set

Public Member Functions

- subroutine [m_set](#) (m, i, j, value)

13.58.1 Detailed Description

interface generic --> set interfaces set : set element of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 344 of file matrix.f90.

13.58.2 Member Function/Subroutine Documentation

13.58.2.1 `subroutine mod_matrix::set::m_set (type(matrix),intent(inout) m, integer,intent(in) i, integer,intent(in) j, real*4,intent(in) value)`

Definition at line 1158 of file matrix.f90.

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

- [matrix.f90](#)

13.59 mod_matrix::spec Interface Reference

interface generic --> spec

Public Member Functions

- `type(vector) m_eig_qr (m, iter_max, meth_qr, eps_gsortho, is_permuted)`

13.59.1 Detailed Description

interface generic --> spec interfaces spec : spec of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

29th of March, 2011

Remarks

Definition at line 353 of file matrix.f90.

13.59.2 Member Function/Subroutine Documentation

13.59.2.1 `type(vector) mod_matrix::spec::m_eig_qr (type(matrix),intent(in) m, integer,intent(in) iter_max, character*(*),intent(in),optional meth_qr, real*4,intent(in),optional eps_gsortho, logical,intent(in),optional is_permuted)`

Definition at line 4171 of file matrix.f90.

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

- [matrix.f90](#)

13.60 mod_vector::sqnorm Interface Reference

interface generic --> sqnorm

Public Member Functions

- `real *4 v_sqLength (v)`

13.60.1 Detailed Description

interface generic --> sqnorm interfaces v_sqrLength : sqnorm 2 of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

27th of March, 2011

Remarks

Definition at line 239 of file vector.f90.

13.60.2 Member Function/Subroutine Documentation

13.60.2.1 real*4 mod_vector::sqnorm::v_sqrLength (type(vector),intent(in) v)

Definition at line 1219 of file vector.f90.

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

- [vector.f90](#)

13.61 mod_vector::sum Interface Reference

interface generic --> sum

Public Member Functions

- real *4 [v_sum](#) (v)

13.61.1 Detailed Description

interface generic --> sum interfaces v_sum : sum of vector

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 185 of file vector.f90.

13.61.2 Member Function/Subroutine Documentation

13.61.2.1 `real*4 mod_vector::sum::v_sum (type(vector),intent(in) v)`

Definition at line 889 of file vector.f90.

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

- [vector.f90](#)

13.62 `mod_matrix::sum` Interface Reference

interface generic --> sum

Public Member Functions

- `real *4 m_sum (m)`

13.62.1 Detailed Description

interface generic --> sum interfaces sum : sum of matrix

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

12th of March, 2011

Remarks

Definition at line 253 of file matrix.f90.

13.62.2 Member Function/Subroutine Documentation

13.62.2.1 `real*4 mod_matrix::sum::m_sum (type(matrix),intent(in) m)`

Definition at line 1653 of file matrix.f90.

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

- [matrix.f90](#)

13.63 `mod_matrix::svd` Interface Reference

interface generic --> svd

Public Member Functions

- `type(t_svd) m_decompsvd` (`m`, `eps`, `iter_max`, `meth_qr`, `eps_gsortho`, `is_permuted`)

13.63.1 Detailed Description

interface generic --> svd interfaces : svd factorisation

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

29th of March, 2011

Remarks

Definition at line 416 of file matrix.f90.

13.63.2 Member Function/Subroutine Documentation

13.63.2.1 `type(t_svd) mod_matrix::svd::m_decompsvd` (`type(matrix)`,`intent(in)`
`m`, `real*4`,`intent(in)`,`optional eps`, `integer`,`intent(in)`,`optional iter_max`,
`character*(*)`,`intent(in)`,`optional meth_qr`, `real*4`,`intent(in)`,`optional eps_gsortho`,
`logical`,`intent(in)`,`optional is_permuted`)

Definition at line 3853 of file matrix.f90.

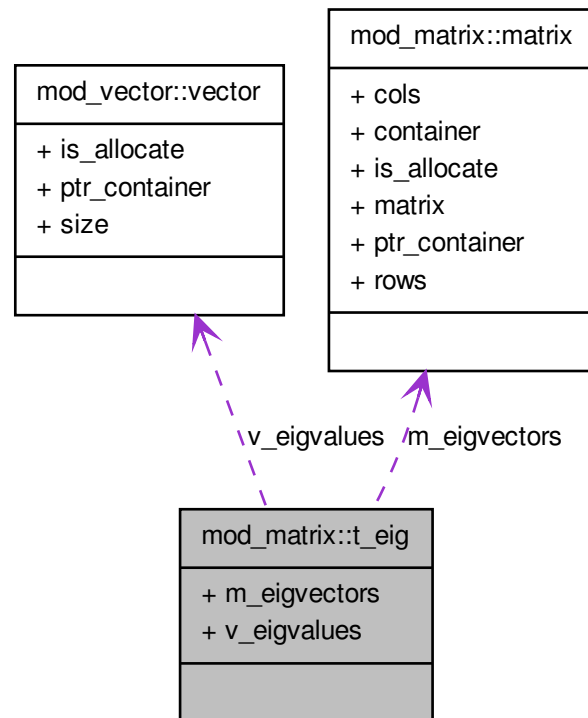
The documentation for this interface was generated from the following file:

- [matrix.f90](#)

13.64 mod_matrix::t_eig Type Reference

type [t_eig](#)

Collaboration diagram for `mod_matrix::t_eig`:



Public Attributes

- type([matrix](#)) `m_eigvectors`
- type([vector](#)) `v_eigvalues`

13.64.1 Detailed Description

type [t_eig](#) eig value and vector(iterativs methods)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

v_eigvalues : type([vector](#)), values
m_eigvectors : type([matrix](#)), eigenvectors

Date

27th of March, 2011

Remarks

Definition at line 129 of file matrix.f90.

13.64.2 Member Data Documentation

13.64.2.1 type(matrix) mod_matrix::t_eig::m_eigvectors

Definition at line 131 of file matrix.f90.

13.64.2.2 type(vector) mod_matrix::t_eig::v_eigvalues

Definition at line 130 of file matrix.f90.

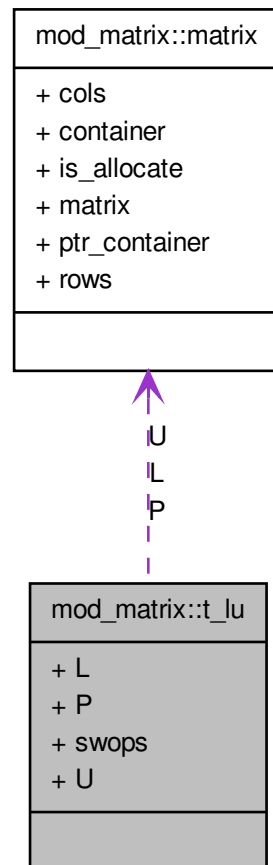
The documentation for this type was generated from the following file:

- [matrix.f90](#)

13.65 mod_matrix::t_lu Type Reference

type [t_lu](#)

Collaboration diagram for `mod_matrix::t_lu`:



Public Attributes

- `type(matrix) L`
- `type(matrix) P`
- integer `swops` = 0
- `type(matrix) U`

13.65.1 Detailed Description

type `t_lu`

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

L : type(matrix), lower matrix
 U : type(matrix), upper matrix
 P : type(matrix), permutation matrix

Date

14th of March, 2011

Remarks

Definition at line 56 of file matrix.f90.

13.65.2 Member Data Documentation**13.65.2.1 type(matrix) mod_matrix::t_lu::L**

Definition at line 57 of file matrix.f90.

13.65.2.2 type(matrix) mod_matrix::t_lu::P

Definition at line 59 of file matrix.f90.

13.65.2.3 integer mod_matrix::t_lu::swops = 0

Definition at line 60 of file matrix.f90.

13.65.2.4 type(matrix) mod_matrix::t_lu::U

Definition at line 58 of file matrix.f90.

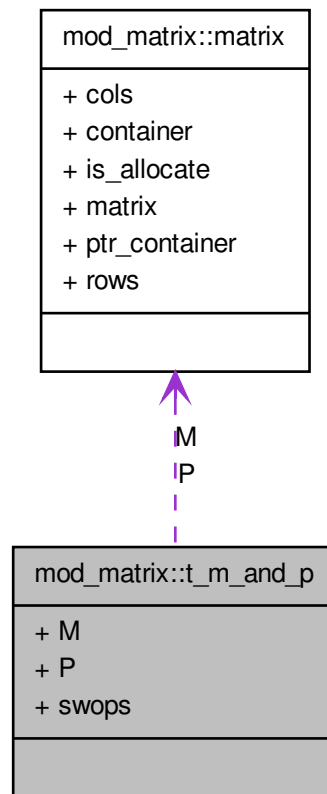
The documentation for this type was generated from the following file:

- [matrix.f90](#)

13.66 mod_matrix::t_m_and_p Type Reference

type [t_m_and_p](#)

Collaboration diagram for `mod_matrix::t_m_and_p`:



Public Attributes

- `type(matrix) M`
- `type(matrix) P`
- integer `swops` = 0

13.66.1 Detailed Description

type `t_m_and_p`

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

M : `type(matrix)`, matrix solution
P : `type(matrix)`, permutation matrix

Date

25th of March, 2011

Remarks

Definition at line 100 of file matrix.f90.

13.66.2 Member Data Documentation**13.66.2.1 type(matrix) mod_matrix::t_m_and_p::M**

Definition at line 101 of file matrix.f90.

13.66.2.2 type(matrix) mod_matrix::t_m_and_p::P

Definition at line 102 of file matrix.f90.

13.66.2.3 integer mod_matrix::t_m_and_p::swops = 0

Definition at line 103 of file matrix.f90.

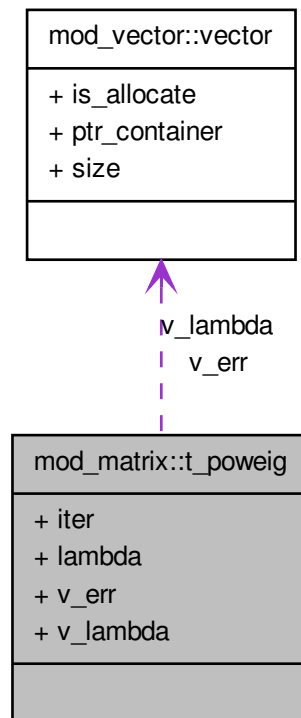
The documentation for this type was generated from the following file:

- [matrix.f90](#)

13.67 mod_matrix::t_poweig Type Reference

type [t_poweig](#)

Collaboration diagram for `mod_matrix::t_poweig`:



Public Attributes

- integer `iter`
- real *4 `lambda`
- type(`vector`) `v_err`
- type(`vector`) `v_lambda`

13.67.1 Detailed Description

type `t_poweig` power eig (iterativs methods)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

lambda :: real*4 , power eig
v_lambda : type(`vector`), eigenvector
v_err : type(`vector`), vector error
iter : integer, number of iteration

Date

26th of March, 2011

Remarks

Definition at line 115 of file matrix.f90.

13.67.2 Member Data Documentation**13.67.2.1 integer mod_matrix::t_poweig::iter**

Definition at line 119 of file matrix.f90.

13.67.2.2 real*4 mod_matrix::t_poweig::lambda

Definition at line 116 of file matrix.f90.

13.67.2.3 type(vector) mod_matrix::t_poweig::v_err

Definition at line 118 of file matrix.f90.

13.67.2.4 type(vector) mod_matrix::t_poweig::v_lambda

Definition at line 117 of file matrix.f90.

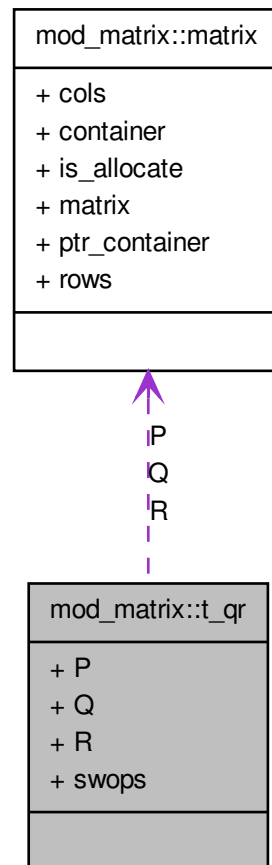
The documentation for this type was generated from the following file:

- [matrix.f90](#)

13.68 mod_matrix::t_qr Type Reference

type [t_qr](#)

Collaboration diagram for `mod_matrix::t_qr`:



Public Attributes

- type([matrix](#)) [P](#)
- type([matrix](#)) [Q](#)
- type([matrix](#)) [R](#)
- integer [swops](#) = 0

13.68.1 Detailed Description

type [t_qr](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

Q : type(matrix), lower matrix
 R : type(matrix), upper matrix
 P : type(matrix), permutation matrix

Date

14th of March, 2011

Remarks

Definition at line 71 of file matrix.f90.

13.68.2 Member Data Documentation**13.68.2.1 type(matrix) mod_matrix::t_qr::P**

Definition at line 74 of file matrix.f90.

13.68.2.2 type(matrix) mod_matrix::t_qr::Q

Definition at line 72 of file matrix.f90.

13.68.2.3 type(matrix) mod_matrix::t_qr::R

Definition at line 73 of file matrix.f90.

13.68.2.4 integer mod_matrix::t_qr::swops = 0

Definition at line 75 of file matrix.f90.

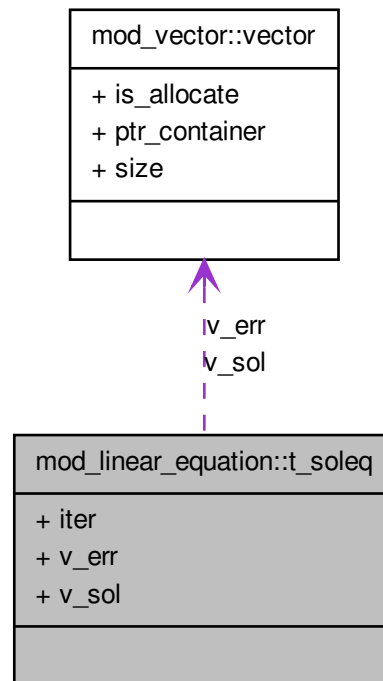
The documentation for this type was generated from the following file:

- [matrix.f90](#)

13.69 mod_linear_equation::t_soleq Type Reference

type [t_soleq](#)

Collaboration diagram for `mod_linear_equation::t_soleq`:



Public Attributes

- integer `iter`
- type(`vector`) `v_err`
- type(`vector`) `v_sol`

13.69.1 Detailed Description

type `t_soleq` soleq for iterativs iterations

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

`v_sol` : type(`vector`), vector solution
`v_err` : type(`vector`), vector error
`iter` : integer, number of iteration

Date

15th of March, 2011

Remarks

Definition at line 34 of file linear_equation.f90.

13.69.2 Member Data Documentation**13.69.2.1 integer mod_linear_equation::t_oleq::iter**

Definition at line 37 of file linear_equation.f90.

13.69.2.2 type(vector) mod_linear_equation::t_oleq::v_err

Definition at line 36 of file linear_equation.f90.

13.69.2.3 type(vector) mod_linear_equation::t_oleq::v_sol

Definition at line 35 of file linear_equation.f90.

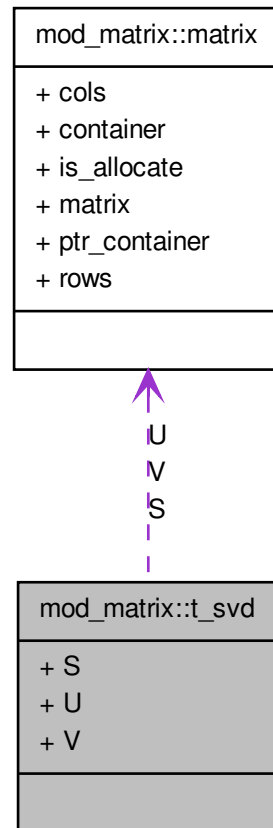
The documentation for this type was generated from the following file:

- [linear_equation.f90](#)

13.70 mod_matrix::t_svd Type Reference

type [t_svd](#)

Collaboration diagram for mod_matrix::t_svd:



Public Attributes

- type(matrix) [S](#)
- type(matrix) [U](#)
- type(matrix) [V](#)

13.70.1 Detailed Description

type [t_svd](#) [S,V,D] with $A = U * S * V'$, $S \geq 0$, $U' * U = I_u$, and $V' * V = I_v$

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

S : type(vector), diagonal matrix (singular values)

V : type(matrix), orthogonal or unitary square matrix $V^* * V = I_v$
 U : type(matrix), orthogonal or unitary square matrix $U^* * U = I_u$

Date

28th of March, 2011

Remarks

Definition at line 86 of file matrix.f90.

13.70.2 Member Data Documentation**13.70.2.1 type(matrix) mod_matrix::t_svd::S**

Definition at line 87 of file matrix.f90.

13.70.2.2 type(matrix) mod_matrix::t_svd::U

Definition at line 89 of file matrix.f90.

13.70.2.3 type(matrix) mod_matrix::t_svd::V

Definition at line 88 of file matrix.f90.

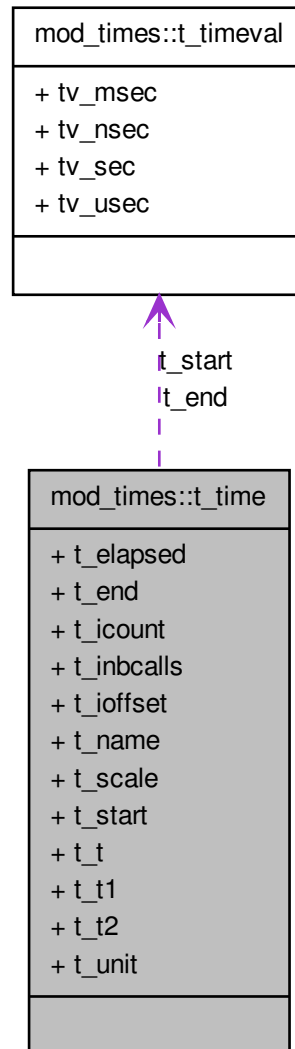
The documentation for this type was generated from the following file:

- [matrix.f90](#)

13.71 mod_times::t_time Type Reference

type [t_time](#)

Collaboration diagram for `mod_times::t_time`:



Public Attributes

- real *4 `t_elapsed`
- type(`t_timeval`) `t_end`
- integer `t_icount`
- integer `t_inbcalls`
- integer `t_ioffset`
- character(len=30) `t_name`
- real *4 `t_scale`

- type([t_timeval](#)) [t_start](#)
- real *4 [t_t](#)
- real *4 [t_t1](#)
- real *4 [t_t2](#)
- character(len=2) [t_unit](#)

13.71.1 Detailed Description

type [t_time](#)

Parameters

t_start : start time type([t_timeval](#))
t_end : end type([t_timeval](#))
t_elapsed : elapsed time
t_t : time
t_t1 : time
t_t2 : time
t_scale : scale time
t_name : time name
t_unit : time unit
t_icount : occurrence time
t_inbcalls : number times of call
t_ioffset : offset
author Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

20th of September, 2010

Remarks

Definition at line 53 of file times.f90.

13.71.2 Member Data Documentation

13.71.2.1 real*4 mod_times::t_time::t_elapsed

Definition at line 56 of file times.f90.

13.71.2.2 type(t_timeval) mod_times::t_time::t_end

Definition at line 55 of file times.f90.

13.71.2.3 integer mod_times::t_time::t_icount

Definition at line 63 of file times.f90.

13.71.2.4 integer mod_times::t_time::t_inbcalls

Definition at line 64 of file times.f90.

13.71.2.5 integer mod_times::t_time::t_ioffset

Definition at line 65 of file times.f90.

13.71.2.6 character(len=30) mod_times::t_time::t_name

Definition at line 61 of file times.f90.

13.71.2.7 real*4 mod_times::t_time::t_scale

Definition at line 60 of file times.f90.

13.71.2.8 type(t_timeval) mod_times::t_time::t_start

Definition at line 54 of file times.f90.

13.71.2.9 real*4 mod_times::t_time::t_t

Definition at line 57 of file times.f90.

13.71.2.10 real*4 mod_times::t_time::t_t1

Definition at line 58 of file times.f90.

13.71.2.11 real*4 mod_times::t_time::t_t2

Definition at line 59 of file times.f90.

13.71.2.12 character(len=2) mod_times::t_time::t_unit

Definition at line 62 of file times.f90.

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

- [times.f90](#)

13.72 mod_times::t_timeval Type Reference

type [t_timeval](#)

Public Attributes

- integer [tv_msec](#)
- integer [tv_nsec](#)
- integer [tv_sec](#)
- integer [tv_usec](#)

13.72.1 Detailed Description

type [t_timeval](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

tv_sec : seconds
tv_msec : milliseconds
tv_usec : microseconds
tv_nsec : nanoseconds

Date

20th of September, 2010

Remarks

Definition at line 28 of file times.f90.

13.72.2 Member Data Documentation

13.72.2.1 integer mod_times::t_timeval::tv_msec

Definition at line 30 of file times.f90.

13.72.2.2 integer mod_times::t_timeval::tv_nsec

Definition at line 32 of file times.f90.

13.72.2.3 integer mod_times::t_timeval::tv_sec

Definition at line 29 of file times.f90.

13.72.2.4 integer mod_times::t_timeval::tv_usec

Definition at line 31 of file times.f90.

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

- [times.f90](#)

13.73 mod_matrix::tril Interface Reference

interface generic --> tril

Public Member Functions

- `type(matrix) m_tril` (m, swap_diag)

13.73.1 Detailed Description

interface generic --> tril interfaces tril : tril (lower matrix)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

28th of March, 2011

Remarks

Definition at line 317 of file matrix.f90.

13.73.2 Member Function/Subroutine Documentation

13.73.2.1 `type(matrix) mod_matrix::tril::m_tril (type(matrix),intent(in) m, integer,intent(in),optional swap_diag)`

Definition at line 2357 of file matrix.f90.

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

- [matrix.f90](#)

13.74 mod_matrix::triu Interface Reference

interface generic --> triu

Public Member Functions

- `type(matrix) m_triu` (m, swap_diag)

13.74.1 Detailed Description

interface generic --> triu interfaces triu : triu (upper matrix)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Date

28th of March, 2011

Remarks

Definition at line 308 of file matrix.f90.

13.74.2 Member Function/Subroutine Documentation

13.74.2.1 type(matrix) mod_matrix::triu::m_triu (type(matrix),intent(in) *m*, integer,intent(in),optional *swap_diag*)

Definition at line 2393 of file matrix.f90.

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

- [matrix.f90](#)

13.75 mod_exception::type_exception Type Reference

type [type_exception](#)

Public Attributes

- integer [e_level](#)
- integer [e_number](#)
- character(len=100) [e_what](#)

13.75.1 Detailed Description

type [type_exception](#)

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

e_number : error number
e_level : error level
e_what : error signification

Date

12th of March, 2011

Remarks

See also

[fml_constants.h](#)

Definition at line 30 of file exception.f90.

13.75.2 Member Data Documentation

13.75.2.1 integer mod_exception::type_exception::e_level

Definition at line 32 of file exception.f90.

13.75.2.2 integer mod_exception::type_exception::e_number

Definition at line 31 of file exception.f90.

13.75.2.3 character(len= 100) mod_exception::type_exception::e_what

Definition at line 33 of file exception.f90.

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

- [exception.f90](#)

13.76 mod_vector::vector Type Reference

real*4 : type of precision : * real*4 for simple precision * real*8 for double precision

Public Attributes

- logical [is_allocate](#) = .false.
- real *4, dimension(:), allocatable [ptr_container](#)
- integer [size](#) = 1;

13.76.1 Detailed Description

real*4 : type of precision : * real*4 for simple precision * real*8 for double precision

Author

Abal-Kassim Cheik Ahamed <akcheik@gmail.com>

Parameters

size : integer (by default =1), vector size
ptr_container : vector container
is_allocate : boolean, verify if the vector is allocate

Date

12th of March, 2011

Remarks

Definition at line 42 of file vector.f90.

13.76.2 Member Data Documentation

13.76.2.1 logical mod_vector::vector::is_allocate = .false.

Definition at line 45 of file vector.f90.

13.76.2.2 real*4,dimension(:),allocatable mod_vector::vector::ptr_container

Definition at line 44 of file vector.f90.

13.76.2.3 integer mod_vector::vector::size = 1;

Definition at line 43 of file vector.f90.

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

- [vector.f90](#)

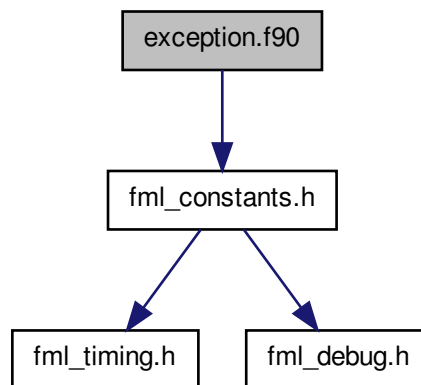
Chapter 14

File Documentation

14.1 exception.f90 File Reference

```
#include "fml_constants.h"
```

Include dependency graph for exception.f90:



Data Types

- type `mod_exception::type_exception`
type `type_exception`

Modules

- module `mod_exception`

module [mod_exception](#)

Functions/Subroutines

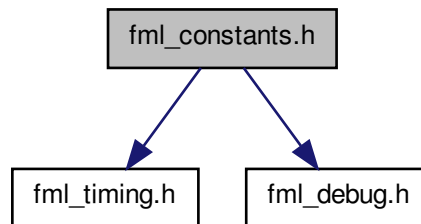
- type(type_exception) [mod_exception::e_error](#) (number_, what_, level_)
subroutine e_print_err(err_)
- subroutine [mod_exception::e_print_err](#) (err_)
subroutine e_print_err(err_)

14.2 fml_constants.h File Reference

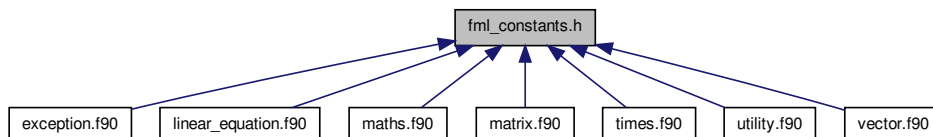
```
#include "fml_timing.h"
```

```
#include "fml_debug.h"
```

Include dependency graph for fml_constants.h:



This graph shows which files directly or indirectly include this file:



Defines

- `#define` [DEBUG_EXCEPTION](#)

- #define [DEBUGPRINT](#)(txtmsg, msg) write(*,*), 'txtmsg', ':', msg
- #define [DEBUGPRINT_MPI](#)(txtmsg, msg, myrank) write(*,*), 'txtmsg', ':', msg, myrank
- #define [len_what_exception](#) 100
- #define [MASTER_ID](#) 0
- #define [mpi_type_precision](#) MPI_REAL
- #define [OMP_NUM_THREADS](#) 2
- #define [p_abs](#) abs
- #define [p_cast](#) dble
- #define [p_dble](#) dble
- #define [p_eps](#) 1.e-16
- #define [p_fmt_file](#) '(f0.8)'
- #define [p_iter_eps](#) 1.e-7
- #define [p_mun](#) -1.0
- #define [p_notcast](#) real
- #define [p_sngl](#) real
- #define [p_sqrt](#) sqrt
- #define [p_un](#) 1.0
- #define [stop_aloc](#) 2001
- #define [stop_arith](#) 100
- #define [stop_array_compatible](#) 202
- #define [stop_array_diag_compatible](#) 203
- #define [stop_array_diverge](#) 301
- #define [stop_array_factorisation](#) 305
- #define [stop_array_indice_exceed](#) 201
- #define [stop_array_positive](#) 304
- #define [stop_array_singular](#) 300
- #define [stop_array_symmetric](#) 302
- #define [stop_dealloc](#) 2002
- #define [stop_div0](#) 101
- #define [stop_err](#) 1101
- #define [stop_method_exist](#) 400
- #define [stop_open_file](#) 3001
- #define [stop_overflow](#) 200
- #define [stop_sqrt](#) 102
- #define [stop_std](#) 1100
- #define [stop_tst](#) 1102
- #define [stop_write_file](#) 3011
- #define [type_precision](#) real*4
- #define [use_type](#)

14.2.1 Define Documentation

14.2.1.1 #define DEBUG_EXCEPTION

Definition at line 78 of file fml_constants.h.

14.2.1.2 #define DEBUGPRINT(txtmsg, msg) write(*,*), 'txtmsg', ':', msg

Definition at line 74 of file fml_constants.h.

14.2.1.3 #define DEBUGPRINT_MPI(*txtmsg*, *msg*, *myrank*) write(*,*), 'txtmsg', ':', msg, myrank

Definition at line 75 of file fml_constants.h.

14.2.1.4 #define len_what_exception 100

Definition at line 72 of file fml_constants.h.

14.2.1.5 #define MASTER_ID 0

Definition at line 81 of file fml_constants.h.

14.2.1.6 #define mpi_type_precision MPI_REAL

Definition at line 44 of file fml_constants.h.

14.2.1.7 #define OMP_NUM_THREADS 2

Definition at line 38 of file fml_constants.h.

14.2.1.8 #define p_abs abs

Definition at line 45 of file fml_constants.h.

14.2.1.9 #define p_cast dble

Definition at line 47 of file fml_constants.h.

14.2.1.10 #define p_dble dble

Definition at line 48 of file fml_constants.h.

14.2.1.11 #define p_eps 1.e-16

Definition at line 53 of file fml_constants.h.

14.2.1.12 #define p_fmt_file '(f0.8)'

Definition at line 55 of file fml_constants.h.

14.2.1.13 #define p_iter_eps 1.e-7

Definition at line 54 of file fml_constants.h.

14.2.1.14 #define p_mun -1.0

Definition at line 52 of file fml_constants.h.

14.2.1.15 #define p_notcast real

Definition at line 50 of file fml_constants.h.

14.2.1.16 #define p_sngl real

Definition at line 49 of file fml_constants.h.

14.2.1.17 #define p_sqrt sqrt

Definition at line 46 of file fml_constants.h.

14.2.1.18 #define p_un 1.0

Definition at line 51 of file fml_constants.h.

14.2.1.19 #define stop_aloc 2001

Definition at line 29 of file fml_constants.h.

14.2.1.20 #define stop_arith 100

Definition at line 8 of file fml_constants.h.

14.2.1.21 #define stop_array_compatible 202

Definition at line 14 of file fml_constants.h.

14.2.1.22 #define stop_array_diag_compatible 203

Definition at line 15 of file fml_constants.h.

14.2.1.23 #define stop_array_diverge 301

Definition at line 18 of file fml_constants.h.

14.2.1.24 #define stop_array_factorisation 305

Definition at line 21 of file fml_constants.h.

14.2.1.25 #define stop_array_indice_exceed 201

Definition at line 13 of file fml_constants.h.

14.2.1.26 #define stop_array_positive 304

Definition at line 20 of file fml_constants.h.

14.2.1.27 #define stop_array_singular 300

Definition at line 17 of file fml_constants.h.

14.2.1.28 #define stop_array_symmetric 302

Definition at line 19 of file fml_constants.h.

14.2.1.29 #define stop_dealloc 2002

Definition at line 30 of file fml_constants.h.

14.2.1.30 #define stop_div0 101

Definition at line 9 of file fml_constants.h.

14.2.1.31 #define stop_err 1101

Definition at line 26 of file fml_constants.h.

14.2.1.32 #define stop_method_exist 400

Definition at line 23 of file fml_constants.h.

14.2.1.33 #define stop_open_file 3001

Definition at line 32 of file fml_constants.h.

14.2.1.34 #define stop_overflow 200

Definition at line 12 of file fml_constants.h.

14.2.1.35 #define stop_sqrt 102

Definition at line 10 of file fml_constants.h.

14.2.1.36 #define stop_std 1100

Definition at line 25 of file fml_constants.h.

14.2.1.37 #define stop_tst 1102

Definition at line 27 of file fml_constants.h.

14.2.1.38 #define stop_write_file 3011

Definition at line 33 of file fml_constants.h.

14.2.1.39 #define type_precision real*4

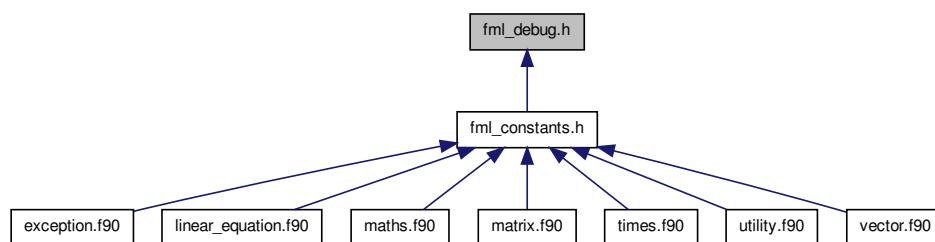
Definition at line 43 of file fml_constants.h.

14.2.1.40 #define use_type

Definition at line 40 of file fml_constants.h.

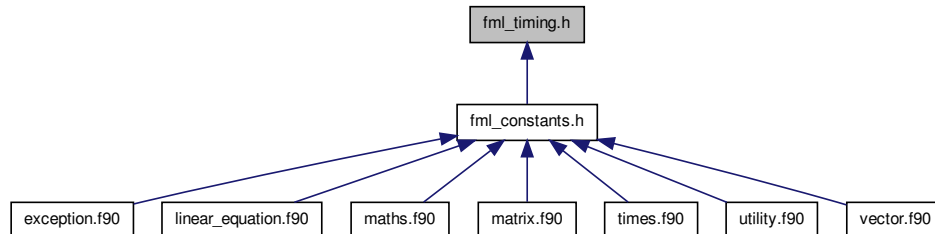
14.3 fml_debug.h File Reference

This graph shows which files directly or indirectly include this file:



14.4 fml_timing.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- `#define` [CLOCKS_PER_SEC](#) 1000
- `#define` [dd_nbCalls](#) 0
- `#define` [dd_OffSet](#) 1

14.4.1 Define Documentation

14.4.1.1 `#define` [CLOCKS_PER_SEC](#) 1000

Definition at line 4 of file `fml_timing.h`.

14.4.1.2 `#define` [dd_nbCalls](#) 0

Definition at line 6 of file `fml_timing.h`.

14.4.1.3 `#define` [dd_OffSet](#) 1

Definition at line 8 of file `fml_timing.h`.

14.5 info.f90 File Reference

Functions/Subroutines

- program [info](#)

14.5.1 Function Documentation

14.5.1.1 program `info` ()

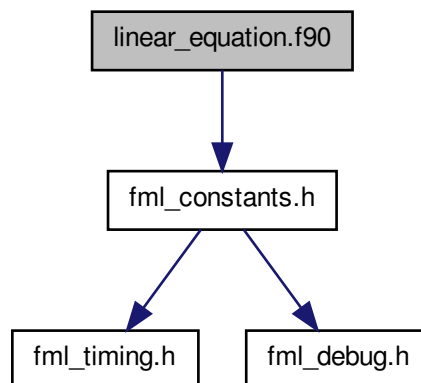
Definition at line 277 of file `info.f90`.

14.6 linear_equation.f90 File Reference

solve linear equation library

```
#include "fml_constants.h"
```

Include dependency graph for linear_equation.f90:



Data Types

- interface `mod_linear_equation::destruct`
interface generic --> destruct linear_equation and derived type
- interface `mod_linear_equation::ilinsolve`
interface generic --> ilinsolve
- interface `mod_linear_equation::leq_getBlock`
interface generic --> leq_getBlock
- interface `mod_linear_equation::linsolve`
interface generic --> linsolve
- type `mod_linear_equation::t_soleq`
type t_soleq

Modules

- module `mod_linear_equation`
module mod_linear_equation

Functions/Subroutines

- type(matrix) [mod_linear_equation::gsb_getBlock_m](#) (m, r_s, r_e, c_s, c_e)
- integer, dimension(size_block) [mod_linear_equation::gsb_index_blocks](#) (tab_block, size_block)
- type(t_soleq) [mod_linear_equation::leq_cg_elt](#) (m, v, v0, eps, iter_max, is_precond)
- type(vector) [mod_linear_equation::leq_Cholesky](#) (m, v, is_permuted)
- type(vector) [mod_linear_equation::leq_gaussJourdan](#) (m, v)
- type(t_soleq) [mod_linear_equation::leq_gaussSeidel_block](#) (m, v, v0, tab_block_i, size_block_i, eps, iter_max, is_precond)
- type(t_soleq) [mod_linear_equation::leq_gaussSeidel_elt](#) (m, v, v0, eps, iter_max, is_precond)
- type(vector) [mod_linear_equation::leq_getBlock_v](#) (v, r_s, r_e)
- type(t_soleq) [mod_linear_equation::leq_gradient_elt](#) (m, v, v0, eps, iter_max, is_precond)
- type(t_soleq) [mod_linear_equation::leq_jacobi_elt](#) (m, v, v0, eps, iter_max, is_precond)
- type(vector) [mod_linear_equation::leq_leastSquare](#) (m, v)
- type(vector) [mod_linear_equation::leq_leastSquare_QR](#) (m, v, meth_qr, eps_gsortho, is_permuted)
- type(vector) [mod_linear_equation::leq_lu](#) (m, v, is_permuted)
- type(vector) [mod_linear_equation::leq_precond_diag](#) (m, v)
- type(vector) [mod_linear_equation::leq_precond_sor](#) (m, v, v0, w_relax)
- type(vector) [mod_linear_equation::leq_pseudoinverse](#) (m, v, eps_svd, eps_chol, iter_max, meth_qr, eps_gsortho, meth_pinv)
- type(vector) [mod_linear_equation::leq_qr](#) (m, v, meth_qr, eps_gsortho, is_permuted)
- type(vector) [mod_linear_equation::leq_solve](#) (m, v, meth_solve, v0, eps_svd, eps_chol, iter_max, meth_qr, meth_pinv, eps_gsortho, is_permuted)
- type(t_soleq) [mod_linear_equation::leq_solve_iter](#) (m, v, meth_solve, v0, eps, iter_max, is_precond, w_relax, tab_block_i, size_block_i)
- type(t_soleq) [mod_linear_equation::leq_sor_elt](#) (m, v, v0, eps, iter_max, w_relax, is_precond)
- type(vector) [mod_linear_equation::leq_thomas](#) (m, v)
- type(vector) [mod_linear_equation::leq_tril](#) (m, v)
- type(vector) [mod_linear_equation::leq_triu](#) (m, v)
- subroutine [mod_linear_equation::m_destruct_soleq](#) (soleq_)

14.6.1 Detailed Description

solve linear equation library

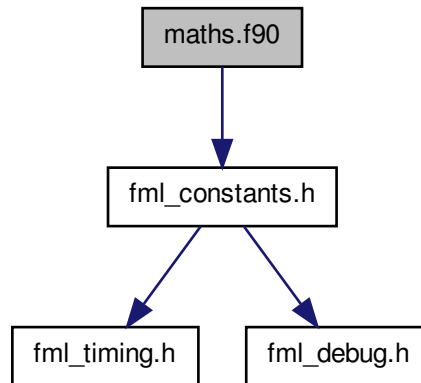
Definition in file [linear_equation.f90](#).

14.7 maths.f90 File Reference

utilities functions

```
#include "fml_constants.h"
```


Include dependency graph for maths.f90:



Modules

- module [mod_maths](#)
module [mod_maths](#)

Functions/Subroutines

- real *4 [mod_maths::math_machine_eps](#) ()
function [math_machine_eps\(filename\)](#) result(res)

Variables

- character(len=100) [mod_maths::math_what_exception](#)
exception signification of maths

14.7.1 Detailed Description

utilities functions

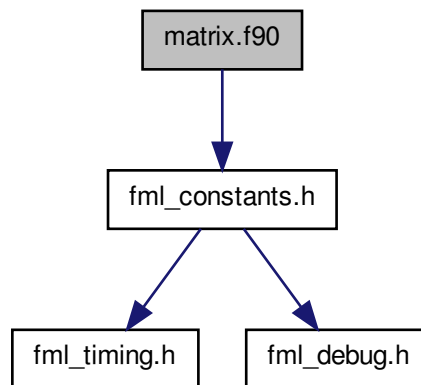
Definition in file [maths.f90](#).

14.8 matrix.f90 File Reference

exception

```
#include "fml_constants.h"
```

Include dependency graph for matrix.f90:



Data Types

- interface `mod_matrix::assignment(=)`
brief interface assignment(=)
- interface `mod_matrix::chol`
interface generic --> chol
- interface `mod_matrix::destruct`
interface generic --> destruct matrix and derived type
- interface `mod_matrix::det`
interface generic --> det
- interface `mod_matrix::diag`
interface generic --> diag
- interface `mod_matrix::get`
interface generic --> get
- interface `mod_matrix::init`
interface generic --> init
- interface `mod_matrix::inv`
interface generic --> inv
- interface `mod_matrix::lu`

interface generic --> lu

- interface `mod_matrix::m_lu`
interface generic --> m_lu
- type `mod_matrix::matrix`
type matrix
- interface `mod_matrix::max`
interface generic --> max
- interface `mod_matrix::min`
interface generic --> min
- interface `mod_matrix::norm`
interface generic --> norm
- interface `mod_matrix::operator(*)`
interface operator()*
- interface `mod_matrix::operator(+)`
interface operator(+)
- interface `mod_matrix::operator(-)`
interface operator(-)
- interface `mod_matrix::operator(.cond.)`
interface operator(.cond.)
- interface `mod_matrix::operator(.det.)`
interface operator(.det.)
- interface `mod_matrix::operator(.inv.)`
interface operator(.inv.)
- interface `mod_matrix::operator(.rank.)`
interface operator(.rank.)
- interface `mod_matrix::operator(.tr.)`
interface operator(.tr.)
- interface `mod_matrix::operator(/)`
interface operator(/)
- interface `mod_matrix::operator(==)`
interface operator(==)
- interface `mod_matrix::pinv`
interface generic --> pinv

- interface `mod_matrix::print`
interface generic --> print
- interface `mod_matrix::qr`
interface generic --> qr
- interface `mod_matrix::random`
interface generic --> random
- interface `mod_matrix::rank`
interface generic --> rank
- interface `mod_matrix::set`
interface generic --> set
- interface `mod_matrix::spec`
interface generic --> spec
- interface `mod_matrix::sum`
interface generic --> sum
- interface `mod_matrix::svd`
interface generic --> svd
- type `mod_matrix::t_eig`
type t_eig
- type `mod_matrix::t_lu`
type t_lu
- type `mod_matrix::t_m_and_p`
type t_m_and_p
- type `mod_matrix::t_poweig`
type t_poweig
- type `mod_matrix::t_qr`
type t_qr
- type `mod_matrix::t_svd`
type t_svd
- interface `mod_matrix::tril`
interface generic --> tril
- interface `mod_matrix::triu`
interface generic --> triu

Modules

- module [mod_matrix](#)
module mod_matrix

Functions/Subroutines

- type(matrix) [mod_matrix::m_add](#) (m1, m2)
- subroutine [mod_matrix::m_affect](#) (m, value)
- type(matrix) [mod_matrix::m_bidiag_low](#) (m)
- type(matrix) [mod_matrix::m_bidiag_up](#) (m)
- real *4 [mod_matrix::m_cond](#) (m)
- type(t_m_and_p) [mod_matrix::m_decompCholesky](#) (m, is_permuted)
- type(t_lu) [mod_matrix::m_decompLU](#) (m, is_permuted)
- type(t_m_and_p) [mod_matrix::m_decompLU_m](#) (m, is_permuted)
- type(t_qr) [mod_matrix::m_decompQR](#) (m, meth_qr, eps_gsortho, is_permuted)
- type(t_qr) [mod_matrix::m_decompQR_GramSchmidt](#) (m, is_permuted)
- type(t_qr) [mod_matrix::m_decompQR_GramSchmidt_Reortho](#) (m, eps, is_permuted)
- type(t_qr) [mod_matrix::m_decompQR_Householder](#) (m, is_permuted)
- type(t_svd) [mod_matrix::m_decompsvd](#) (m, eps, iter_max, meth_qr, eps_gsortho, is_permuted)
- type(vector) [mod_matrix::m_decompsvd_s](#) (m, eps, iter_max, meth_qr, eps_gsortho, is_permuted)
- subroutine [mod_matrix::m_destruct](#) (m)
- subroutine [mod_matrix::m_destruct_lu](#) (lu_)
- subroutine [mod_matrix::m_destruct_m_and_p](#) (m_and_p_)
- subroutine [mod_matrix::m_destruct_qr](#) (qr_)
- subroutine [mod_matrix::m_destruct_t_eig](#) (t_eig_)
- subroutine [mod_matrix::m_destruct_t_poweig](#) (t_poweig_)
- subroutine [mod_matrix::m_destruct_t_svd](#) (t_svd_)
- real *4 [mod_matrix::m_det](#) (m, meth_det, is_permuted)
- real *4 [mod_matrix::m_det_chol](#) (m, is_permuted)
- real *4 [mod_matrix::m_det_gaussj](#) (m)
- real *4 [mod_matrix::m_det_lu](#) (m, is_permuted)
- real *4 [mod_matrix::m_det_lu_all](#) (m, is_permuted)
- type(vector) [mod_matrix::m_diag](#) (m, i)
- type(matrix) [mod_matrix::m_div_scalar](#) (m, alpha)
- type(t_eig) [mod_matrix::m_eig_deflation](#) (m, v0, eps, iter_max)
- type(vector) [mod_matrix::m_eig_qr](#) (m, iter_max, meth_qr, eps_gsortho, is_permuted)
- type(matrix) [mod_matrix::m_extract](#) (m, r_lbound, r_ubound, c_lbound, c_ubound)
- real *4 [mod_matrix::m_get](#) (m, i, j)
- real *4, dimension(m%rows, m%cols) [mod_matrix::m_get_m](#) (m)
- real *4, dimension(m%rows) [mod_matrix::m_getCol](#) (m, j)
- real *4, dimension(m%cols) [mod_matrix::m_getRow](#) (m, i)
- integer [mod_matrix::m_getSize](#) (m)
- integer [mod_matrix::m_getSizeCols](#) (m)
- integer [mod_matrix::m_getSizeRows](#) (m)
- type(matrix) [mod_matrix::m_identity](#) (n)
- subroutine [mod_matrix::m_init](#) (m, rows_, cols_)
- subroutine [mod_matrix::m_init_fromfile](#) (m, filename, unit)
- type(matrix) [mod_matrix::m_inverse_gaussj](#) (m)

- logical `mod_matrix::m_isEqual` (m1, m2)
- logical `mod_matrix::m_isEqual_scalar` (m, val)
- logical `mod_matrix::m_isSymmetric` (m)
- type(vector) `mod_matrix::m_matrixTOvector` (m)
- real *4 `mod_matrix::m_max` (m)
- real *4 `mod_matrix::m_maxCol` (m, j)
- real *4 `mod_matrix::m_maxRow` (m, i)
- real *4 `mod_matrix::m_min` (m)
- real *4 `mod_matrix::m_minCol` (m, j)
- subroutine `mod_matrix::m_minit_value` (m, value)
- real *4 `mod_matrix::m_minRow` (m, i)
- type(matrix) `mod_matrix::m_minus` (m1, m2)
- integer `mod_matrix::m_nbnegative` (m)
- integer `mod_matrix::m_nbnegativeCol` (m, j)
- integer `mod_matrix::m_nbnegativeRow` (m, i)
- integer `mod_matrix::m_nbpositive` (m)
- integer `mod_matrix::m_nbpositiveCol` (m, j)
- integer `mod_matrix::m_nbpositiveRow` (m, i)
- integer `mod_matrix::m_nbzeros` (m)
- integer `mod_matrix::m_nbzerosCol` (m, j)
- integer `mod_matrix::m_nbzerosRow` (m, i)
- real *4 `mod_matrix::m_norm` (m, type_norm)
- type(t_m_and_p) `mod_matrix::m_permut` (m, is_permuted)
- type(t_m_and_p) `mod_matrix::m_permut_col` (m, is_permuted)
- type(matrix) `mod_matrix::m_pinv` (m, eps_svd, eps_chol, iter_max, meth_qr, eps_gsortho, meth_pinv)
- type(t_poweig) `mod_matrix::m_pow_eig` (m, v0, eps, iter_max)
- subroutine `mod_matrix::m_print` (m)
- subroutine `mod_matrix::m_print_lu_tofile` (lu_decomp, filename, unit, status, position)
- subroutine `mod_matrix::m_print_m_and_p_tofile` (m_and_p_decomp, filename, unit, status, position)
- subroutine `mod_matrix::m_print_qr_tofile` (qr_decomp, filename, unit, status, position)
- subroutine `mod_matrix::m_print_tofile` (m, filename, unit, status, position)
- type(matrix) `mod_matrix::m_prod_mat` (m1, m2)
- type(matrix) `mod_matrix::m_prod_scalar1` (alpha, m)
- type(matrix) `mod_matrix::m_prod_scalar2` (m, alpha)
- type(vector) `mod_matrix::m_prod_vec1` (m, v)
- type(vector) `mod_matrix::m_prod_vec2` (v, m)
- type(vector) `mod_matrix::m_prod_vec_c` (m, v)
- type(matrix) `mod_matrix::m_pseudoinv_chol` (m, eps_chol)
- type(matrix) `mod_matrix::m_pseudoinv_svd` (m, eps_svd, iter_max, meth_qr, eps_gsortho)
- integer `mod_matrix::m_rank` (m, tol_rank, meth_rk, eps_svd, iter_max, meth_qr, eps_gsortho, is_permuted)
- integer `mod_matrix::m_rank_gaussj` (m)
- integer `mod_matrix::m_rank_svd` (m, tol_rank, eps_svd, iter_max, meth_qr, eps_gsortho, is_permuted)
- subroutine `mod_matrix::m_resize` (m, rows_, cols_)
- subroutine `mod_matrix::m_set` (m, i, j, value)
- subroutine `mod_matrix::m_setsub` (m, r_lbound, r_ubound, c_lbound, c_ubound, m_sub)
- real *4 `mod_matrix::m_sum` (m)
- real *4 `mod_matrix::m_sumCol` (m, j)
- real *4 `mod_matrix::m_sumRow` (m, i)

- real *4 `mod_matrix::m_trace` (m)
- type(matrix) `mod_matrix::m_trans` (m)
- type(matrix) `mod_matrix::m_tril` (m, swap_diag)
- type(matrix) `mod_matrix::m_triu` (m, swap_diag)
- type(matrix) `mod_matrix::m_zeros` (rows_, cols_)
- type(matrix) `mod_matrix::mc_bidiag_low` (d, l, size_n)
- type(matrix) `mod_matrix::mc_bidiag_up` (d, u, size_n)
- type(matrix) `mod_matrix::mc_diag` (vect_data)
- subroutine `mod_matrix::mc_diagDominant` (m, n, low, high)
- subroutine `mod_matrix::mc_diagDominantSymmetric` (m, n, low, high)
- subroutine `mod_matrix::mc_random` (m, low, high)
- type(matrix) `mod_matrix::mc_tridiag` (d, u, l, size_n)

Variables

- integer, parameter `mod_matrix::frobenius` = -1
- character(len=100) `mod_matrix::m_what_exception`

brief exception signification of matrix

14.8.1 Detailed Description

exception matrix library

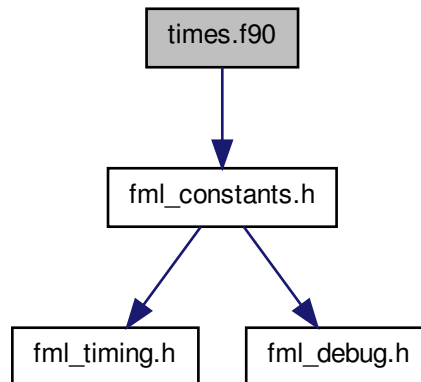
Definition in file `matrix.f90`.

14.9 times.f90 File Reference

times

```
#include "fml_constants.h"
```

Include dependency graph for times.f90:



Data Types

- type `mod_times::t_time`
type `t_time`
- type `mod_times::t_timeval`
type `t_timeval`

Modules

- module `mod_times`
module `mod_times`

Functions/Subroutines

- subroutine `mod_times::ft_begin` (`timing_`)
subroutine `ft_begin(timing_)`
- subroutine `mod_times::ft_create_time` (`timing_`, `name_`, `inbcalls_`, `ioffset_`)
subroutine `ft_create_time(timing_,name_,inbcalls_,ioffset_)`
- subroutine `mod_times::ft_create_time_copy` (`timing_`, `timingcopy_`)
subroutine `ft_create_time_copy(timing_,timingcopy_)`
- subroutine `mod_times::ft_create_time_init` (`timing_`)

subroutine ft_create_time_init(timing_)

- subroutine [mod_times::ft_end](#) (timing_)

subroutine ft_end(timing_)

- subroutine [mod_times::ft_gettimeofday](#) (t_timeval_)

subroutine ft_gettimeofday(t_timeval_)

- subroutine [mod_times::ft_print](#) (timing_, myrank)

subroutine ft_print(timing_, myrank)

- subroutine [mod_times::ft_reset](#) (timing_)

subroutine ft_reset(timing_)

14.9.1 Detailed Description

times

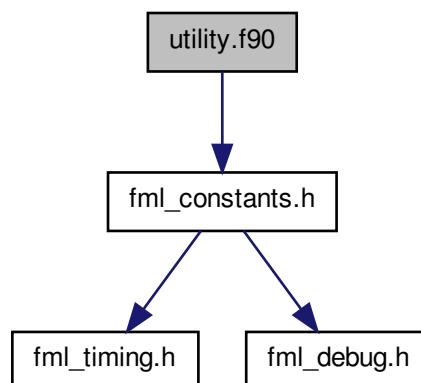
Definition in file [times.f90](#).

14.10 utility.f90 File Reference

utilities functions

```
#include "fml_constants.h"
```

Include dependency graph for utility.f90:



Modules

- module [mod_utility](#)

module [mod_utility](#)

Functions/Subroutines

- logical [mod_utility::u_is_exist_file](#) (filename)

function [u_is_exist_file](#)(filename) result(res)

- integer [mod_utility::u_nblne](#) (filename)

function [u_nblne](#)(filename) result(res)

Variables

- character(len=100) [mod_utility::u_what_exception](#)

exception signification of utility

14.10.1 Detailed Description

utilities functions

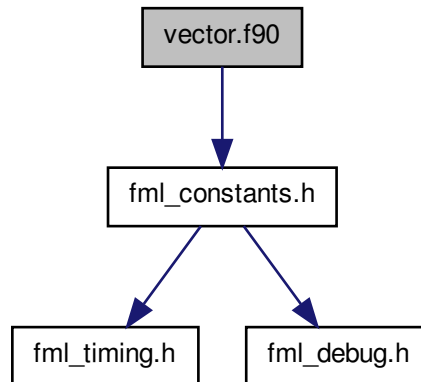
Definition in file [utility.f90](#).

14.11 [vector.f90](#) File Reference

vector library

```
#include "fml_constants.h"
```

Include dependency graph for vector.f90:



Data Types

- interface `mod_vector::abs`
interface generic --> abs
- interface `mod_vector::add`
interface generic --> add
- interface `mod_vector::assignment(=)`
brief interface assignment(=)
- interface `mod_vector::destruct`
interface generic --> destruct vector
- interface `mod_vector::dot`
interface generic --> dot
- interface `mod_vector::get`
interface generic --> get
- interface `mod_vector::init`
interface generic --> init
- interface `mod_vector::max`
interface generic --> max
- interface `mod_vector::min`
interface generic --> min

- interface `mod_vector::norm`
interface generic --> norm
- interface `mod_vector::operator(*)`
interface operator()*
- interface `mod_vector::operator(+)`
interface operator(+)
- interface `mod_vector::operator(-)`
interface operator(-)
- interface `mod_vector::operator(.cross.)`
interface operator(.cross.)
- interface `mod_vector::operator(.dot.)`
interface operator(.dot.)
- interface `mod_vector::operator(.inv.)`
interface operator(.inv.)
- interface `mod_vector::operator(.len.)`
interface operator(.len.)
- interface `mod_vector::operator(.norm.)`
interface operator(.norm.)
- interface `mod_vector::operator(.sqnorm.)`
interface operator(.sqnorm.)
- interface `mod_vector::operator(/)`
interface operator(/)
- interface `mod_vector::operator(==)`
interface operator(==)
- interface `mod_vector::print`
interface generic --> print
- interface `mod_vector::random`
interface generic --> random
- interface `mod_vector::set`
interface generic --> set
- interface `mod_vector::sqnorm`
interface generic --> sqnorm
- interface `mod_vector::sum`

```
interface generic --> sum
```

- type `mod_vector::vector`

*real*4 : type of precision : * real*4 for simple precision * real*8 for double precision*

Modules

- module `mod_vector`

module mod_vector

Functions/Subroutines

- real *4 `mod_vector::v_abs` (v)
- type(vector) `mod_vector::v_add` (v1, v2)
- subroutine `mod_vector::v_add_val_end` (v, val)
- subroutine `mod_vector::v_affect` (v, value)
- type(vector) `mod_vector::v_axpby` (alpha, x, beta, y)
- type(vector) `mod_vector::v_cross` (v1, v2)
- subroutine `mod_vector::v_destruct` (v)
- type(vector) `mod_vector::v_div_scalar` (v, alpha)
- real *4 `mod_vector::v_dot` (v1, v2)
- type(vector) `mod_vector::v_extract` (v, l_bound, u_bound)
- real *4 `mod_vector::v_get` (v, i)
- real *4, dimension(v%size) `mod_vector::v_get_v` (v)
- subroutine `mod_vector::v_init` (v, size_v)
- subroutine `mod_vector::v_init_fromfile` (v, filename, unit)
- subroutine `mod_vector::v_init_value` (v, value)
- type(vector) `mod_vector::v_inverse` (v)
- logical `mod_vector::v_isEqual` (v1, v2)
- logical `mod_vector::v_isEqual_scalar` (v, val)
- real *4 `mod_vector::v_length` (v)
- real *4 `mod_vector::v_max` (v)
- real *4 `mod_vector::v_min` (v)
- type(vector) `mod_vector::v_minus` (v1, v2)
- integer `mod_vector::v_nbnegative` (v)
- integer `mod_vector::v_nbpositive` (v)
- integer `mod_vector::v_nbzeros` (v)
- real *4 `mod_vector::v_norm` (v, type_norm)
- subroutine `mod_vector::v_normalize` (v)
- type(vector) `mod_vector::v_ones` (size_v)
- subroutine `mod_vector::v_print` (v)
- subroutine `mod_vector::v_print_c` (v)
- subroutine `mod_vector::v_print_c_tofile` (v, filename, unit, status, position)
- subroutine `mod_vector::v_print_tofile` (v, filename, unit, status, position)
- real *4 `mod_vector::v_prod` (v)
- type(vector) `mod_vector::v_prod_scalar1` (v, alpha)
- type(vector) `mod_vector::v_prod_scalar2` (alpha, v)

- type(vector) [mod_vector::v_prod_vec](#) (v1, v2)
- subroutine [mod_vector::v_resize](#) (v, size_v)
- subroutine [mod_vector::v_set](#) (v, i, value)
- integer [mod_vector::v_size](#) (v)
- real *4 [mod_vector::v_sqrLength](#) (v)
- real *4 [mod_vector::v_sum](#) (v)
- type(vector) [mod_vector::v_zeros](#) (size_v)
- subroutine [mod_vector::vc_random](#) (v, low, high)

Variables

- integer, parameter [mod_vector::infty](#) = 0
- character(len=100) [mod_vector::v_what_exception](#)

14.11.1 Detailed Description

vector library

Definition in file [vector.f90](#).

Example Documentation

15.1 leq_exception.f90

This example show an example of catch exception with the `linear_equation` module. More details about this example.

[illegible]


```

64 print*, ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> solve :: Gradient element "
65 x=ilinsolve(A,b,meth_solve='gradelt')
66 write(*,'(A5)',advance='no')"* x="; call print(x%v_sol);
67 write(*,'(A24,I3)')"# number of iterations=", x%iter
68 write(*,'(A9)',advance='no')"# A*x-b="; call print(A*x%v_sol-b);
69 print*; !newline
70 print*, ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> solve :: Conjugate Gradient element
    "
71 x=ilinsolve(A,b,meth_solve='cgelt')
72 write(*,'(A5)',advance='no')"* x="; call print(x%v_sol);
73 write(*,'(A24,I3)')"# number of iterations=", x%iter
74 write(*,'(A9)',advance='no')"# A*x-b="; call print(A*x%v_sol-b);
75 print*; !newline
76 print*, ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> solve :: Gauss-Seidel block "
77 !see function prototype for more information (example block composition)
78 x=ilinsolve(A,b,meth_solve='gsblock',eps=p_notcast(1.e-5))
79 write(*,'(A5)',advance='no')"* x="; call print(x%v_sol);
80 write(*,'(A24,I3)')"# number of iterations=", x%iter
81 write(*,'(A9)',advance='no')"# A*x-b="; call print(A*x%v_sol-b);
82 print*; !newline
83
84 print*, "***** Solve linear equation pr
econditionning"
85 print*; !newline
86 print*, ">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> solve :: Conjugate Gradient element
    preconditionning"
87 x=ilinsolve(A,b,meth_solve='cgelt',is_precond=.true.)
88 write(*,'(A5)',advance='no')"* x="; call print(x%v_sol);
89 write(*,'(A24,I3)')"# number of iterations=", x%iter
90 write(*,'(A9)',advance='no')"# A*x-b="; call print(A*x%v_sol-b);
91 print*; !newline
92
93 call destruct(A)      ! destruct the matrix A   (don't forget to destruct the matrix
    )
94 call destruct(b)
95 call destruct(x)
96 end program leq_isolve

```

15.3 leq_solve.f90

This example show how use `linear_equation` module for solve linear equation. We will try to solve $Ax=b$ by classical solves methods (not iteratives method) More details about this example.

```

1
16 program leq_solve
17 #include "fml_constants.h"
18 use mod_linear_equation      ! use linear_equation module
19 implicit none
20
21 !***** declaration
22 integer, parameter :: size_ab=4
23 type(matrix) :: A !main square matrix
24 type(vector) :: b !second member
25 type(vector) :: x !solution
26 !others
27 type(matrix) :: LowA, UpA, TriDiag
28
29 !***** body
30 call init(A,size_ab,size_ab); !init:=m_init
31 call init(b,size_ab); !init:=v_init
32 !initialize A by random values between 1.0 and 10.0 (dominant diagonal)
33 call mc_diagDominant(A,size_ab,low=p_notcast(1.0),high=p_notcast(6.5))
34 !initialize b by random values between 1.0 and 5.0
35 call random(b,low=p_notcast(1.0),high=p_notcast(6.5))

```


[illegible]

15.4 leq_solve_times.f90

This part show how measure cpu times with times module. More details about this example.

```

1
15 program leq_solve_times
16 #include "fml_constants.h"
17 use mod_linear_equation      ! use linear_equation module
18 use mod_times
19 implicit none
20
21 !***** declaration
22 integer :: size_ab=100 !by default
23 type(matrix) :: A !main square matrix
24 type(vector) :: b !second member
25 type(vector) :: x !solution
26
27 !decalre time variable
28 type(t_time) :: time_gj, time_ls, time_lsqr
29 type(t_time) :: time_pinvsvd, time_pinvchol
30 type(t_time) :: time_lu, time_qr, time_chol
31 !*****init times
32 character(len=30) time_name;
33 time_name='time_gj'; call ft_create_time(time_gj,time_name,0,0)
34 time_name='time_ls'; call ft_create_time(time_ls,time_name,0,0)
35 time_name='time_lsqr'; call ft_create_time(time_lsqr,time_name,0,0)
36 time_name='time_pinvchol'; call ft_create_time(time_pinvchol,time_name,0,0)
37
37 time_name='time_lu'; call ft_create_time(time_lu,time_name,0,0)
38 time_name='time_qr'; call ft_create_time(time_qr,time_name,0,0)

```



```

104 call ft_print(time_pinvchol);
105 call ft_print(time_lu);
106 call ft_print(time_qr);
107 call ft_print(time_chol);
108
109 call destruct(A) ! destruct the matrix A (don't forget to destruct the matrix
    )
110 call destruct(b)
111 call destruct(x)
112 end program leq_solve_times

```

15.5 matrix_analysis.f90

This example show how use matricial analysis with the matrix module. More details about this example.

```

1
14 program matrix_analysis
15
16 use mod_matrix      ! use matrix module
17
18 #include "fml_constants.h"
19 implicit none
20 !***** declaration
21 integer :: m2_rows = 4, m2_cols = 5 !size of m1
22 integer :: m3_rows = 5, m3_cols = 5 !size of m1
23 ! declaration of matrix m2
24 type(matrix) :: m2;
25 ! declaration of matrix m3
26 type(matrix) :: m3;
27 ! declaration of matrix m_res
28 type(matrix) :: m_res;
29
30 !***** body
31 !indication: p_notcast is defined in fml_constants.h (adapt the format)
32 ! init of a matrix m3
33 call init(m3,m3_rows,m3_cols); !init:=m_init
34 ! init of a matrix m2
35 call init(m2,m2_rows,m2_cols); !init:=m_init
36
37 !initialize m1 by random values between 1.0 and 10.0
38 call random(m2,low=p_notcast(1.0),high=p_notcast(10.0)) !random:=vc_random
39 !initialize m3 by random values between 2.0 and 6.0
40 call mc_diagDominant(m3,m3_rows,low=p_notcast(1.0),high=p_notcast(1.5)) !dominant diagonal
41
42 print*, "***** display initial data"
43 print*; !newline
44
45 print*, "m2=";
46 call print(m2); !print m2 (print:=m_print)
47 print*, "m3=";
48 call print(m3); !print m3 (print:=m_print)
49
50 print*, "***** matrix mathematics proprieties"
51
52 print*, ">>>> determinant of m3 by gauss-jourdan=", det(m3) !or .det.m3 fo gauss-jourdan method
53 !determinant by lu decomposition
54 print*, ">>>> determinant of m3 by lu decomposition=", det(m3, meth_det='lu')
55
56 print*, ">>>> rank of m2 by gauss-jourdan=", rank(m2) !or .rank.m2 fo gauss-jourdan method
57 !determinant by svd decomposition

```



```

59 print*, ">>>>>>>>>>>>>>>> pseudo-inverse of m2="
60 m_res=pinv(m2,meth_pinv='svd') ! pseudo-inverse by svd
61 print*, "**** svd::pinv(m2,meth_pinv='svd')="
62 call print(m_res);
63 m_res=pinv(m2) ! or pinv(m2,meth_pinv='chol') pseudo-inverse by cholesky
64 print*, "**** svd::pinv(m2,meth_pinv='svd')="
65 call print(m_res);
66 print*;
67 print*, "**** pseudo-inverse verification by Moore-Penrose conditions"
68 print*, "## m2.m2^+m2 = m2"
69 call print(m2*m_res*m2)
70 print*, "m2=";
71 call print(m2);
72 print*;print*, "## m2^+m2 m2^+= m2^+"
73 call print(m_res*m2*m_res)
74 print*, "m2^+=";
75 call print(m_res);
76 print*;print*, "## (m2.m2^+)^* = m2.m2^+"
77 call print(.tr.(m2*m_res))
78 print*, "m2.m2^+=";
79 call print(m2*m_res)
80 print*;print*, "## (m2^+.m2)^* = m2^+.m2"
81 call print(.tr.(m_res*m2))
82 print*, "m2^+.m2="
83 call print(m_res*m2)
84
85
86 print*
87 print*;print*, "... .. deallocate matrix"
88 call destruct(m2) ! destruct the matrix m2
89 call destruct(m3) ! destruct the matrix m3
90 call destruct(m_res) ! destruct the matrix m_res
91 print*;
92 end program matrix_analysis2
```

15.7 matrix_cholesky.f90

This example show how use cholesky decomposition with the matrix module. More details about this example.

```

1
14 program matrix_cholesky
15 #include "fml_constants.h"
16 use mod_matrix      ! use matrix module
17 implicit none
18
19 !***** declaration
20 integer, parameter :: m_size = 4 !size of m(A_size x m_size)
21 ! declaration of matrix A
22 type(matrix) :: A;
23 ! declaration of matrix B
24 type(matrix) :: B;
25 !type contains cholesky matrix and the permutation if it's exist
26 type(t_m_and_p) :: decomp_cholesky;
27
28 !***** body
29 !initialize A by random values between 1.0 and 10.0
30 ! (dominant diagonal, positive definite)
31 call mc_diagDominantSymmetric(A,m_size,low=p_notcast(1.0),high=p_notcast(6.5))
    !dominant diagonal
32 !init B matrix
33 call init(B,m_size,m_size)
34 !initialize B by random values between 1.0 and 10.0
35 call random(B,low=p_notcast(1.0),high=p_notcast(6.5)) !

```



```

40 call print(poweig%v_lambda)
41 print*, ">>>>>>>>>>>>>>> iteration errors = "
42 call print(poweig%v_err)
43 print*; !newline
44
45 print*, "***** eigen value and vector (deflation method), not w
    ork now"
46 eigvalvect=m_eig_deflation(A,eps=p_notcast(1.e-5),iter_max=500) !deflation met
    hod
47 print*, ">>>>>>>>>>>>>>> eigenvalues = "
48 call print(eigvalvect%v_eigenvalues)
49 print*, ">>>>>>>>>>>>>>> eigenvectors = "
50 call print(eigvalvect%m_eigvectors)
51
52 print*; !newline
53
54 call destruct(A) ! destruct the matrix A (don't forget to destruct the matri
    x)
55 call destruct(poweig)
56 call destruct(eigvalvect)
57 end program matrix_eigenvalvect

```

15.9 matrix_exception.f90

This is an example of how an exception can be caught with the matrix module. More details about this example.

```

1
14 program matrix_exception
15 #include "fml_constants.h"
16 use mod_matrix      ! use matrix module
17 implicit none
18
19 !***** declaration
20 integer, parameter :: m1_rows = 6, m1_cols = 4 !size of m1
21 integer, parameter :: m2_rows = 4, m2_cols = 5 !size of m1
22 ! declaration of matrix m1
23 type(matrix) :: m1;
24 ! declaration of matrix m2
25 type(matrix) :: m2;
26 ! declaration of matrix m_res
27 type(matrix) :: m_res;
28 !***** body
29
30 ! init of a matrix m1
31 call init(m1,m1_rows,m1_cols);      !init:=m_init
32 ! init of a matrix m2
33 call init(m2,m2_rows,m2_cols);      !init:=m_init
34
35 !initialize m1 by random values between 1.0 and 10.0
36 !p_notcast is defined on fml_constants.h
37 call random(m1,low=p_notcast(1.0),high=p_notcast(10.0))      !random:=mc_random
38
39 print*, "***** try to addition m1+m2"
40 m_res=m1+m2 !must throw an exception because m1%size != m2%size
41
42 call destruct(m1) ! destruct the matrix m1 (don't forget to destruct the matr
43 ix)
44 call destruct(m2) ! destruct the matrix m2 (don't forget to destruct the matr
45 ix)
46 end program matrix_exception

```


[illegible]

15.12 matrix_manip.f90

This is an example of how to use basic manipulate function of the matrix module. More details about this example.

```

1
14 program matrix_manip
15
16 use mod_matrix          ! use matrix module
17 #include "fml_constants.h"
18 implicit none
19 !***** declaration
20 integer, parameter :: m1_rows = 4, m1_cols = 4 !size of m1
21 integer, parameter :: m2_rows = 4, m2_cols = 5 !size of m2
22 integer, parameter :: m3_rows = 4, m3_cols = 5 !size of m3
23 ! declaration of matrix m1
24 type(matrix) :: m1;
25 ! declaration of matrix m2
26 type(matrix) :: m2;
27 ! declaration of matrix m3
28 type(matrix) :: m3;
29 ! declaration of matrix m4
30 type(matrix) :: m4;
31 ! declaration of matrix m res

```

```

32  type(matrix) :: m_res;
33
34  !***** body
35  !indication:  p_notcast is defined in fml_constants.h (adapt the format)
36  ! init of a matrix m3
37  call init(m3,m3_rows,m3_cols);    !init:=m_init
38  ! init of a matrix m2
39  call init(m2,m2_rows,m2_cols);    !init:=m_init
40
41  !initialize m1 by random values between 1.0 and 10.0
42  call random(m2,low=p_notcast(1.0),high=p_notcast(10.0))  !random:=vc_random
43  !initialize m3 by random values between 2.0 and 6.0
44  call random(m3,low=p_notcast(2.0),high=p_notcast(6.0))  !random:=vc_random
45
46  m1=m_identity(m1_rows)  !identity matrix
47  m4=mc_diag(vect_data=(/p_notcast(1.0),p_notcast(3.0),p_notcast(4.0),p_notcast(-
    5.0),p_notcast(8.0)/))
48
49
50  print*, "***** display initial data"
51  print*; !newline
52
53  print*, "m1=";
54  call print(m1);  !print m1  (print:=m_print)
55  print*, "m2=";
56  call print(m2);  !print m2  (print:=m_print)
57  print*, "m3=";
58  call print(m3);  !print m3  (print:=m_print)
59  print*, "m4=";
60  call print(m4);  !print m4  (print:=m_print)
61
62  print*, "***** basic functions"
63  m_res=p_notcast(2.0)*m2 !multiplication by a scalar
64  print*, "2*m3=";
65  call print(m_res)
66
67  m_res=m3/p_notcast(2.0) !division by a scalar
68  print*, "m3/2=";
69  call print(m_res)
70
71  m_res=m2+m3 !addition
72  print*, "m2+m3=";
73  call print(m_res)
74
75  m_res=m2-m3 !soustraction
76  print*, "m2-m3=";
77  call print(m_res)
78
79  m_res=m1*m3
80  print*, "m1*m3=";
81  call print(m_res)
82
83  print*, "***** matrix mathematics prop-
    erties"
84  print*, "diag(m2)=";
85  call print(diag(m2))
86  print*, "diag(m2,2)=";
87  call print(diag(m2,2))
88  print*, "diag(m2,-2)=";
89  call print(diag(m2,-2))
90  print*, "triu(m2)=";
91  call print(triu(m2)) !upper triangular matrix of m2
92  print*, "tril(m2)=";
93  call print(tril(m2)) !lower triangular matrix of m2
94  print*, "tril(m2,-1)=";
95  call print(tril(m2,-1)) !lower triangular matrix of m2 (left translation)

```

```

96
97 !sauv m2
98 call print(m2,"matrix_m2.dat");
99 print*;*print*, "... .. deallocate matrix"
100 call destruct(m1) ! destruct the matrix m1 (don't forget to destruct the matr
ix)
101 call destruct(m2) ! destruct the matrix m2
102 call destruct(m3) ! destruct the matrix m3
103 call destruct(m4) ! destruct the matrix m4
104 call destruct(m_res) ! destruct the matrix m_res
105 print*;*
106 end program matrix_manip

```

15.13 matrix_qr.f90

This example show how use qr decomposition with the matrix module. More details about this example.

[illegible]

[illegible]

15.15 vector_exception.f90

This is an example of how an exception can caught with the vector module. More details about this example.

```

14 program vector_exception
15 #include "fml_constants.h"
16 use mod_vector      ! use vector module
17 implicit none
18
19 !***** declaration
20 integer, parameter :: v1_size = 5 !size of v1
21 integer, parameter :: v2_size = 6 !size of v1
22 ! declaration of vector v1
23 type(vector) :: v1;
24 ! declaration of vector v2
25 type(vector) :: v2;
26 ! declaration of vector v_res
27 type(vector) :: v_res;
28 !***** body
29
30 ! init of a vector v1
31 call init(v1,v1_size);      !init:=v_init
32 ! init of a vector v2
33 call init(v2,v2_size);      !init:=v_init
34
35 !initialize v1 by random values between 1.0 and 10.0
36 call random(v1,low=p_notcast(1.0),high=p_notcast(10.0))      !random:=vc_random
37
38 print*, "***** try to addition v1+v2"
39 v_res=v1+v2 !must throw an exception because v1%size != v2%size
40
41 call destruct(v1) ! destruct the vector v1 (don't forget to destruct the vect
   or)
42 call destruct(v2) ! destruct the vector v2 (don't forget to destruct the vect
   or)
43 end program vector_exception

```

15.16 vector_init.f90

This is an example of how to use basic initialize function of the vector module. More details about this example.

```

1
14 program vector_init
15 #include "fml_constants.h"
16 use mod_vector      ! use vector module
17 implicit none
18
19 !***** declaration
20 integer, parameter :: v1_size = 5 !size of v1
21 ! declaration of vector v1
22 type(vector) :: v1;
23
24 !***** body
25
26 ! init of a vector v1
27 call init(v1,v1_size);      !init:=v_init
28 !initialize v1 by random values between 1.0 and 10.0
29 call random(v1,low=p_notcast(1.0),high=p_notcast(10.0)) !random:=vc_random
30
31 print*, "***** display data"
32
33 write(*,fmt='(A5)',advance='no') "* v1=";  !(advance='no' => without newline li
    ne)
34 call print(v1); !print v1 (print:=v_print)
35 print*, ">>>>>>>>>size info: size v1=", v1%size
36 call destruct(v1) ! destruct the vector v1 (don't forget to destruct the vect
    or)
37 end program vector_init

```

15.17 vector_manip.f90

This is an example of how to use basic manipulate function of the vector module. More details about this example.

```

1
14 program vector_manip
15
16 use mod_vector      ! use vector module
17 #include "fml_constants.h"
18 implicit none
19 !***** declaration
20 integer, parameter :: v1_size = 6 !size of v1
21 integer, parameter :: v2_size = 6 !size of v1
22 integer :: i
23 ! declaration of vector v1
24 type(vector) :: v1;
25 ! declaration of vector v2
26 type(vector) :: v2;
27 ! declaration of vector v_res
28 type(vector) :: v_res;
29
30 !***** body
31 !indication: p_notcast is defined in fml_constants.h (adapt the format)
32 ! init of a vector v1
33 call init(v1,v1_size);      !init:=v_init
34 ! init of a vector v2
35 call init(v2,v2_size);      !init:=v_init
36

```



```

37 !initialize v1 by random values between 1.0 and 10.0
38 call random(v1,low=p_notcast(1.0),high=p_notcast(10.0)) !random:=vc_random
39 !initialize v2
40 do i=1,v2_size
41     call set(v2, i, p_notcast(i**2)); !v2(i)=i^2    (set:=v_set)
42 end do
43
44 print*, "***** display initial data"
45 print*; !newline
46 write(*,fmt='(A3)',advance='no') "v1=";    !(advance='no' => without newline line
47 )
48 call print(v1); !print v1 (print:=v_print)
49 write(*,fmt='(A3)',advance='no') "v2=";    !(advance='no' => without newline line
50 )
51 call print(v2); !print v2 (print:=v_print)
52
53 print*, "***** basic functions"
54 print*, ">>>>>>>>>nb negative value of v1: =", v_nbnegative(v1); !get:=v_ge
55 t
56 print*, ">>>>>>>>>new nb positive value of v1: =", v_nbpositive(v1); !get:=
57 v_get
58 call set(v1,1,-get(v1,1)) ! v1(1)=-v1(1)    (set:=v_set, get:=v_get)
59 write(*,fmt='(A9)',advance='no') "* new v1=";    !(advance='no' => without newlin
60 e line)
61 call print(v1); !print v1 (print:=v_print)
62 print*, ">>>>>>>>>get value: v1(3)=", get(v1,3); !get:=v_get
63 print*, ">>>>>>>>>new nb negative value of v1: =", v_nbnegative(v1); !get:=
64 v_get
65 print*, ">>>>>>>>>new nb positive value of v1: =", v_nbpositive(v1); !get:=
66 v_get
67 print*, ">>>>>>>>>new nb positive value of v1: =", v_nbpositive(v1); !get:=
68 v_get
69 print*, ">>>>>>>>>nb nil value of v1: =", v_nbzeros(v1);
70 print*, ">>>>>>>>>nb nil value of v1: =", v_nbzeros(v1);
71 print*, ">>>>>>>>>min value of v1: =", min(v1); !min:=v_min
72 print*, ">>>>>>>>>max value of v1: =", max(v1); !max:=v_max
73 print*, ">>>>>>>>>min value of v2: =", min(v2); !min:=v_min
74 print*, ">>>>>>>>>max value of v2: =", max(v2); !max:=v_max
75
76 print*, "***** vector mathematics prop
77 erties"
78 print*, ">>>>>>>>>norm 2 of v1: =", .norm.v1; !or norm(v1), norm(v1,2)
79 print*, ">>>>>>>>>norm 2 of v2: =", norm(v2,2); !norm:=v_norm
80 print*, ">>>>>>>>>norm 10 of v2: =", norm(v2,type_norm=10); !norm:=v_norm
81
82 print*, ">>>>>>>>>norm inf ty of v1: =", norm(v2,inf ty); !norm:=v_norm
83 print*;
84 print*, ">>>>>>>>>v1 dot v2 : =", v1.dot.v2 ! or dot(v1,v2)    dot:=v_dot
85 print*;
86 print*, ">>>>>>>>>v1 - v2 : =",
87 v_res=v1-v2 ! or v_minus(v1,v2)
88 call print(v_res);
89 print*, ">>>>>>>>>v1 + v2 : =",
90 v_res=v1+v2 ! or v_add(v1,v2)
91 call print(v_res);
92 print*, ">>>>>>>>>2*v2=";
93 v_res=p_notcast(2.0)*v2 ! or v_prod_scalar2(2,v2)
94 call print(v_res);
95 print*, ">>>>>>>>>v1*5=";
96 v_res=v1*p_notcast(5.0) ! or v_prod_scalar1(v1,5)
97 call print(v_res);
98 print*, ">>>>>>>>>v2/10=";
99 v_res=v1/p_notcast(10.0) ! or v_div_scalar(v2,10)
100 call print(v_res);
101
102 !sauv v2

```

```
94  call print(v2,"vector_v2.dat");
95  print*;print*, "... ... .. deallocate vector"
96  call destruct(v1) ! destruct the vector v1 (don't forget to destruct the vect
    or)
97  call destruct(v2) ! destruct the vector v2
98  call destruct(v_res) ! destruct the vector v_res
99  print*;
100 end program vector_manip
```