

RcdMathLib\_doc

Generated by Doxygen 1.8.16



<b>1 RcdMathLib Documentation</b>	<b>1</b>
1.1 RcdMathLib in a nutshell	1
1.2 Download and use the RcdMathLib	1
1.3 The quickest start	1
1.4 Structure	2
1.4.1 Linear Algebra	2
1.4.2 Non-Linear Algebra	3
1.4.3 Localization	3
1.4.4 examples	3
1.4.5 doc	3
1.5 Further information	3
<b>2 Getting started</b>	<b>5</b>
2.1 Downloading RcdMathLib code	5
2.2 Compiling RcdMathLib	5
2.2.1 Setting up a toolchain for full-fledged devices	5
2.2.2 Setting up a toolchain for resource-limited devices	6
2.2.3 The build system for full-fledged devices	6
2.2.4 The build system for resource-limited devices	7
2.2.5 Building and executing an example for resource-limited devices	8
<b>3 Creating an application</b>	<b>9</b>
3.1 Creating an application for full-fledged devices	9
3.2 Creating an application for resource-limited devices	9
3.3 The main function	9
3.4 The application's Makefile	10
3.4.1 The minimal Makefile	10
3.4.2 Including modules	10
<b>4 Creating modules</b>	<b>11</b>
4.1 The general structure	11
4.2 Module dependencies	12
<b>5 Module Index</b>	<b>13</b>
5.1 Modules	13
<b>6 Data Structure Index</b>	<b>15</b>
6.1 Data Structures	15
<b>7 File Index</b>	<b>17</b>
7.1 File List	17
<b>8 Module Documentation</b>	<b>21</b>
8.1 BASIC_OPERATIONS	21
8.1.1 Detailed Description	21

8.2 DISTANCE_BASED . . . . .	22
8.2.1 Detailed Description . . . . .	22
8.3 EXAMPLES . . . . .	23
8.3.1 Detailed Description . . . . .	23
8.4 LINEAR_ALGEBRA . . . . .	24
8.4.1 Detailed Description . . . . .	24
8.5 LOCALIZATION . . . . .	25
8.5.1 Detailed Description . . . . .	25
8.6 MAGNETIC_BASED . . . . .	26
8.6.1 Detailed Description . . . . .	26
8.7 MATRIX_DECOMPOSITIONS . . . . .	27
8.7.1 Detailed Description . . . . .	27
8.8 NON_LINEAR_ALGEBRA . . . . .	28
8.8.1 Detailed Description . . . . .	28
8.9 OPTIMIZATION . . . . .	29
8.9.1 Detailed Description . . . . .	29
8.10 POSITION_ALGOS . . . . .	30
8.10.1 Detailed Description . . . . .	30
8.11 POSITION_OPTIMIZATION . . . . .	31
8.11.1 Detailed Description . . . . .	31
8.12 POS_ALGOS_COMMON . . . . .	32
8.12.1 Detailed Description . . . . .	32
8.13 PSEUDO_INVERSE . . . . .	33
8.13.1 Detailed Description . . . . .	33
8.14 SOLVE_LINEAR_EQUATIONS . . . . .	34
8.14.1 Detailed Description . . . . .	34
8.15 SOLVE_NON_LINEAR_EQUATIONS . . . . .	35
8.15.1 Detailed Description . . . . .	35
8.16 UTILITIES . . . . .	36
8.16.1 Detailed Description . . . . .	36
<b>9 Data Structure Documentation</b>	<b>37</b>
9.1 matrix_dim_t Struct Reference . . . . .	37
9.1.1 Detailed Description . . . . .	37
<b>10 File Documentation</b>	<b>39</b>
10.1 combinatorics.c File Reference . . . . .	39
10.1.1 Detailed Description . . . . .	39
10.1.2 Function Documentation . . . . .	39
10.1.2.1 combinatorics_get_next_without_rep() . . . . .	39
10.1.2.2 combinatorics_init() . . . . .	40
10.2 combinatorics.h File Reference . . . . .	40
10.2.1 Detailed Description . . . . .	41

10.2.2 Function Documentation . . . . .	41
10.2.2.1 combinatorics_get_next_without_rep() . . . . .	41
10.2.2.2 combinatorics_init() . . . . .	42
10.3 damped_newton_raphson.c File Reference . . . . .	42
10.3.1 Detailed Description . . . . .	43
10.3.2 Function Documentation . . . . .	43
10.3.2.1 damped_newton_raphson() . . . . .	43
10.3.2.2 get_damped_norm() . . . . .	44
10.3.2.3 get_delta_x() . . . . .	45
10.4 damped_newton_raphson.h File Reference . . . . .	45
10.4.1 Detailed Description . . . . .	46
10.4.2 Function Documentation . . . . .	46
10.4.2.1 damped_newton_raphson() . . . . .	46
10.4.2.2 get_damped_norm() . . . . .	47
10.4.2.3 get_delta_x() . . . . .	48
10.5 dist_based_fi.c File Reference . . . . .	48
10.5.1 Detailed Description . . . . .	49
10.5.2 Function Documentation . . . . .	49
10.5.2.1 dist_based_f_i() . . . . .	49
10.5.2.2 dist_based_fi() . . . . .	49
10.6 dist_based_fi.h File Reference . . . . .	50
10.6.1 Detailed Description . . . . .	50
10.6.2 Function Documentation . . . . .	50
10.6.2.1 dist_based_f_i() . . . . .	50
10.6.2.2 dist_based_fi() . . . . .	51
10.7 dist_based_jacobian.c File Reference . . . . .	51
10.7.1 Detailed Description . . . . .	52
10.7.2 Function Documentation . . . . .	52
10.7.2.1 dist_based_jacobian_get_J() . . . . .	52
10.7.2.2 dist_based_jacobian_get_J_mul_s() . . . . .	53
10.7.2.3 dist_based_jacobian_get_JTf() . . . . .	53
10.7.2.4 dist_based_jacobian_get_JTJ() . . . . .	54
10.8 dist_based_jacobian.h File Reference . . . . .	54
10.8.1 Detailed Description . . . . .	55
10.8.2 Function Documentation . . . . .	55
10.8.2.1 dist_based_jacobian_get_J() . . . . .	55
10.8.2.2 dist_based_jacobian_get_J_mul_s() . . . . .	55
10.8.2.3 dist_based_jacobian_get_JTf() . . . . .	56
10.8.2.4 dist_based_jacobian_get_JTJ() . . . . .	57
10.9 dist_based_position.c File Reference . . . . .	57
10.9.1 Detailed Description . . . . .	57
10.9.2 Function Documentation . . . . .	58

10.9.2.1 dist_based_get_absolute_error()	58
10.9.2.2 dist_based_get_distance_to_anchor()	58
10.10 dist_based_position.h File Reference	59
10.10.1 Detailed Description	59
10.10.2 Function Documentation	59
10.10.2.1 dist_based_get_absolute_error()	59
10.10.2.2 dist_based_get_distance_to_anchor()	60
10.11 distance_based_test.c File Reference	60
10.11.1 Detailed Description	60
10.11.2 Function Documentation	61
10.11.2.1 distance_based_test()	61
10.12 distance_based_test.h File Reference	61
10.12.1 Detailed Description	61
10.12.2 Function Documentation	61
10.12.2.1 distance_based_test()	62
10.13 DOP.c File Reference	62
10.13.1 Detailed Description	62
10.13.2 Function Documentation	62
10.13.2.1 get_PDOP()	62
10.14 DOP.h File Reference	63
10.14.1 Detailed Description	63
10.14.2 Function Documentation	63
10.14.2.1 get_PDOP()	63
10.15 fsolve.c File Reference	64
10.15.1 Detailed Description	64
10.15.2 Function Documentation	64
10.15.2.1 fsolve()	65
10.16 fsolve.h File Reference	66
10.16.1 Detailed Description	66
10.16.2 Enumeration Type Documentation	66
10.16.2.1 NON_LIN_ALGORITHM	66
10.16.3 Function Documentation	67
10.16.3.1 fsolve()	67
10.17 fsolve_test.c File Reference	68
10.17.1 Detailed Description	68
10.17.2 Function Documentation	68
10.17.2.1 get_non_lin_sys_f1_()	68
10.17.2.2 get_non_lin_sys_f2_()	69
10.17.2.3 get_non_lin_sys_f3_()	69
10.17.2.4 get_non_lin_sys_J1_()	70
10.17.2.5 get_non_lin_sys_J2_()	71
10.17.2.6 get_non_lin_sys_J3_()	71

10.18 fsolve_test.h File Reference	72
10.18.1 Detailed Description	72
10.19 givens_test.c File Reference	72
10.19.1 Detailed Description	73
10.20 givens_test.h File Reference	73
10.20.1 Detailed Description	73
10.21 householder_test.c File Reference	73
10.21.1 Detailed Description	74
10.22 householder_test.h File Reference	74
10.22.1 Detailed Description	74
10.23 levenberg_marquardt.c File Reference	74
10.23.1 Detailed Description	75
10.23.2 Function Documentation	75
10.23.2.1 opt_levenberg_marquardt()	75
10.23.2.2 opt_levenberg_marquardt_correction()	76
10.23.2.3 opt_levenberg_marquardt_get_mu0()	77
10.24 levenberg_marquardt.h File Reference	78
10.24.1 Detailed Description	78
10.24.2 Function Documentation	78
10.24.2.1 opt_levenberg_marquardt()	78
10.24.2.2 opt_levenberg_marquardt_get_mu0()	79
10.25 loc_gauss_newton.c File Reference	80
10.25.1 Detailed Description	80
10.25.2 Function Documentation	80
10.25.2.1 loc_gauss_newton()	81
10.26 loc_gauss_newton.h File Reference	82
10.26.1 Detailed Description	82
10.26.2 Function Documentation	82
10.26.2.1 loc_gauss_newton()	82
10.27 loc_levenberg_marquardt.c File Reference	83
10.27.1 Detailed Description	84
10.27.2 Function Documentation	84
10.27.2.1 loc_levenberg_marquardt()	84
10.27.2.2 loc_levenberg_marquardt_correction()	86
10.27.2.3 loc_levenberg_marquardt_get_JTJ_mu2_l()	87
10.27.2.4 loc_levenberg_marquardt_get_mu0()	87
10.28 loc_levenberg_marquardt.h File Reference	88
10.28.1 Detailed Description	88
10.28.2 Function Documentation	89
10.28.2.1 loc_levenberg_marquardt()	89
10.28.2.2 loc_levenberg_marquardt_correction()	90
10.28.2.3 loc_levenberg_marquardt_get_JTJ_mu2_l()	91

10.28.2.4 loc_levenberg_marquardt_get_mu0()	92
10.29 lu_decomp.c File Reference	93
10.29.1 Detailed Description	93
10.29.2 Function Documentation	93
10.29.2.1 lu_decomp()	94
10.30 lu_decomp.h File Reference	94
10.30.1 Detailed Description	95
10.30.2 Function Documentation	95
10.30.2.1 lu_decomp()	95
10.31 lu_decomp_test.c File Reference	96
10.31.1 Detailed Description	96
10.32 lu_decomp_test.h File Reference	96
10.32.1 Detailed Description	96
10.33 magnetic_based_fi.c File Reference	97
10.33.1 Detailed Description	97
10.33.2 Function Documentation	97
10.33.2.1 magnetic_based_f_i()	97
10.34 magnetic_based_fi.h File Reference	98
10.34.1 Detailed Description	98
10.34.2 Function Documentation	98
10.34.2.1 magnetic_based_f_i()	98
10.35 magnetic_based_jacobian.c File Reference	99
10.35.1 Detailed Description	99
10.35.2 Function Documentation	100
10.35.2.1 magnetic_based_jacobian_get_J()	100
10.35.2.2 magnetic_based_jacobian_get_J_mul_s()	100
10.35.2.3 magnetic_based_jacobian_get_JTf()	101
10.35.2.4 magnetic_based_jacobian_get_JTJ()	101
10.36 magnetic_based_jacobian.h File Reference	102
10.36.1 Detailed Description	102
10.36.2 Function Documentation	102
10.36.2.1 magnetic_based_jacobian_get_J()	103
10.36.2.2 magnetic_based_jacobian_get_J_mul_s()	103
10.36.2.3 magnetic_based_jacobian_get_JTf()	104
10.36.2.4 magnetic_based_jacobian_get_JTJ()	104
10.37 magnetic_based_position.c File Reference	105
10.37.1 Detailed Description	105
10.37.2 Function Documentation	106
10.37.2.1 magnetic_based_get_absolute_error()	106
10.37.2.2 magnetic_based_get_distances()	106
10.37.2.3 magnetic_based_get_distances_to_anchors()	107
10.37.2.4 magnetic_based_get_magnetic_field()	107

10.37.2.5 magnetic_based_get_magnetic_field_vec()	108
10.37.2.6 magnetic_based_get_r()	109
10.37.2.7 magnetic_based_preprocessing_get_position()	109
10.38 magnetic_based_position.h File Reference	110
10.38.1 Detailed Description	111
10.38.2 Function Documentation	111
10.38.2.1 magnetic_based_get_absolute_error()	111
10.38.2.2 magnetic_based_get_distances()	112
10.38.2.3 magnetic_based_get_distances_to_anchors()	112
10.38.2.4 magnetic_based_get_magnetic_field()	113
10.38.2.5 magnetic_based_get_magnetic_field_vec()	113
10.38.2.6 magnetic_based_get_r()	114
10.38.2.7 magnetic_based_preprocessing_get_position()	114
10.39 magnetic_based_test.c File Reference	115
10.39.1 Detailed Description	115
10.40 magnetic_based_test.h File Reference	115
10.40.1 Detailed Description	116
10.41 matrix.c File Reference	116
10.41.1 Detailed Description	118
10.41.2 Function Documentation	119
10.41.2.1 matrix_add()	119
10.41.2.2 matrix_add_to_diag()	119
10.41.2.3 matrix_clear()	120
10.41.2.4 matrix_copy()	120
10.41.2.5 matrix_flex_part_print()	121
10.41.2.6 matrix_flex_print()	121
10.41.2.7 matrix_get_abs_max_elem_and_index_in_part_column()	122
10.41.2.8 matrix_get_abs_max_elem_in_column()	123
10.41.2.9 matrix_get_abs_max_elem_in_part_column()	124
10.41.2.10 matrix_get_column_vec()	124
10.41.2.11 matrix_get_diag_mat()	125
10.41.2.12 matrix_get_diag_mat_new()	125
10.41.2.13 matrix_get_frob_norm()	126
10.41.2.14 matrix_get_inv_low_triangular()	126
10.41.2.15 matrix_get_inv_upp_triangular()	127
10.41.2.16 matrix_get_low_triangular()	127
10.41.2.17 matrix_get_max_elem_in_column()	127
10.41.2.18 matrix_get_max_elem_in_part_column()	128
10.41.2.19 matrix_get_part_column_vec()	129
10.41.2.20 matrix_get_rank()	129
10.41.2.21 matrix_get_two_norm()	130
10.41.2.22 matrix_get_upp_triangular()	130

10.41.2.23 <code>matrix_in_place_transpose()</code> . . . . .	131
10.41.2.24 <code>matrix_init()</code> . . . . .	131
10.41.2.25 <code>matrix_mul()</code> . . . . .	132
10.41.2.26 <code>matrix_mul_col_vec_row_vec()</code> . . . . .	132
10.41.2.27 <code>matrix_mul_scalar()</code> . . . . .	133
10.41.2.28 <code>matrix_mul_scalar_vec_matr()</code> . . . . .	133
10.41.2.29 <code>matrix_mul_vec()</code> . . . . .	134
10.41.2.30 <code>matrix_part_copy()</code> . . . . .	134
10.41.2.31 <code>matrix_part_mul()</code> . . . . .	135
10.41.2.32 <code>matrix_part_mul_scalar_vec_matr()</code> . . . . .	136
10.41.2.33 <code>matrix_part_print()</code> . . . . .	137
10.41.2.34 <code>matrix_part_swap_rows()</code> . . . . .	137
10.41.2.35 <code>matrix_print()</code> . . . . .	138
10.41.2.36 <code>matrix_read()</code> . . . . .	138
10.41.2.37 <code>matrix_set_diag_elements()</code> . . . . .	139
10.41.2.38 <code>matrix_sub()</code> . . . . .	139
10.41.2.39 <code>matrix_swap_rows()</code> . . . . .	140
10.41.2.40 <code>matrix_trans_mul_itself()</code> . . . . .	140
10.41.2.41 <code>matrix_trans_mul_vec()</code> . . . . .	141
10.41.2.42 <code>matrix_transpose()</code> . . . . .	141
10.41.2.43 <code>matrix_vec_mul_matr()</code> . . . . .	142
10.41.2.44 <code>matrix_write()</code> . . . . .	142
10.42 <code>matrix.h</code> File Reference . . . . .	143
10.42.1 Detailed Description . . . . .	146
10.42.2 Macro Definition Documentation . . . . .	146
10.42.2.1 <code>M_PI</code> . . . . .	146
10.42.2.2 <code>MACHEPS</code> . . . . .	146
10.42.2.3 <code>matrix_t</code> . . . . .	146
10.42.3 Function Documentation . . . . .	146
10.42.3.1 <code>matrix_add()</code> . . . . .	146
10.42.3.2 <code>matrix_add_to_diag()</code> . . . . .	147
10.42.3.3 <code>matrix_clear()</code> . . . . .	147
10.42.3.4 <code>matrix_copy()</code> . . . . .	148
10.42.3.5 <code>matrix_flex_part_print()</code> . . . . .	148
10.42.3.6 <code>matrix_flex_print()</code> . . . . .	149
10.42.3.7 <code>matrix_get_abs_max_elem_and_index_in_part_column()</code> . . . . .	150
10.42.3.8 <code>matrix_get_abs_max_elem_in_column()</code> . . . . .	150
10.42.3.9 <code>matrix_get_abs_max_elem_in_part_column()</code> . . . . .	151
10.42.3.10 <code>matrix_get_column_vec()</code> . . . . .	151
10.42.3.11 <code>matrix_get_diag_mat()</code> . . . . .	152
10.42.3.12 <code>matrix_get_diag_mat_new()</code> . . . . .	152
10.42.3.13 <code>matrix_get_frob_norm()</code> . . . . .	153

10.42.3.14	<a href="#">matrix_get_inv_low_triang()</a>	153
10.42.3.15	<a href="#">matrix_get_inv_upp_triang()</a>	154
10.42.3.16	<a href="#">matrix_get_low_triang()</a>	154
10.42.3.17	<a href="#">matrix_get_max_elem_in_column()</a>	155
10.42.3.18	<a href="#">matrix_get_max_elem_in_part_column()</a>	155
10.42.3.19	<a href="#">matrix_get_part_column_vec()</a>	156
10.42.3.20	<a href="#">matrix_get_rank()</a>	156
10.42.3.21	<a href="#">matrix_get_two_norm()</a>	157
10.42.3.22	<a href="#">matrix_get_upp_triang()</a>	158
10.42.3.23	<a href="#">matrix_in_place_transpose()</a>	158
10.42.3.24	<a href="#">matrix_init()</a>	158
10.42.3.25	<a href="#">matrix_mul()</a>	159
10.42.3.26	<a href="#">matrix_mul_col_vec_row_vec()</a>	160
10.42.3.27	<a href="#">matrix_mul_scalar()</a>	160
10.42.3.28	<a href="#">matrix_mul_scalar_vec_matr()</a>	161
10.42.3.29	<a href="#">matrix_mul_vec()</a>	162
10.42.3.30	<a href="#">matrix_part_copy()</a>	162
10.42.3.31	<a href="#">matrix_part_mul()</a>	163
10.42.3.32	<a href="#">matrix_part_mul_scalar_vec_matr()</a>	164
10.42.3.33	<a href="#">matrix_part_print()</a>	165
10.42.3.34	<a href="#">matrix_part_swap_rows()</a>	165
10.42.3.35	<a href="#">matrix_print()</a>	166
10.42.3.36	<a href="#">matrix_read()</a>	166
10.42.3.37	<a href="#">matrix_set_diag_elements()</a>	167
10.42.3.38	<a href="#">matrix_sub()</a>	167
10.42.3.39	<a href="#">matrix_swap_rows()</a>	168
10.42.3.40	<a href="#">matrix_trans_mul_itself()</a>	168
10.42.3.41	<a href="#">matrix_trans_mul_vec()</a>	169
10.42.3.42	<a href="#">matrix_transpose()</a>	169
10.42.3.43	<a href="#">matrix_vec_mul_matr()</a>	170
10.42.3.44	<a href="#">matrix_write()</a>	170
10.43	<a href="#">matrix_test.c File Reference</a>	171
10.43.1	<a href="#">Detailed Description</a>	171
10.44	<a href="#">matrix_test.h File Reference</a>	172
10.44.1	<a href="#">Detailed Description</a>	172
10.45	<a href="#">modified_gauss_newton.c File Reference</a>	172
10.45.1	<a href="#">Detailed Description</a>	173
10.45.2	<a href="#">Function Documentation</a>	173
10.45.2.1	<a href="#">modified_gauss_newton()</a>	173
10.46	<a href="#">modified_gauss_newton.h File Reference</a>	174
10.46.1	<a href="#">Detailed Description</a>	174
10.46.2	<a href="#">Function Documentation</a>	175

10.46.2.1 <code>modified_gauss_newton()</code> . . . . .	175
10.47 <code>moore_penrose_pinv_test.c</code> File Reference . . . . .	176
10.47.1 Detailed Description . . . . .	176
10.48 <code>moore_penrose_pinv_test.h</code> File Reference . . . . .	176
10.48.1 Detailed Description . . . . .	176
10.49 <code>moore_penrose_pseudo_inverse.c</code> File Reference . . . . .	177
10.49.1 Detailed Description . . . . .	177
10.49.2 Function Documentation . . . . .	177
10.49.2.1 <code>moore_penrose_get_pinv()</code> . . . . .	177
10.49.2.2 <code>moore_penrose_pinv_compute_print()</code> . . . . .	178
10.50 <code>moore_penrose_pseudo_inverse.h</code> File Reference . . . . .	178
10.50.1 Detailed Description . . . . .	179
10.50.2 Function Documentation . . . . .	179
10.50.2.1 <code>moore_penrose_get_pinv()</code> . . . . .	179
10.50.2.2 <code>moore_penrose_pinv_compute_print()</code> . . . . .	180
10.51 <code>multipath_algo_own_norm_distr_test.c</code> File Reference . . . . .	180
10.51.1 Detailed Description . . . . .	181
10.52 <code>multipath_algo_own_norm_distr_test.h</code> File Reference . . . . .	181
10.52.1 Detailed Description . . . . .	181
10.53 <code>multipath_dist_detection_mitigation.c</code> File Reference . . . . .	182
10.53.1 Detailed Description . . . . .	182
10.53.2 Function Documentation . . . . .	183
10.53.2.1 <code>get_exact_distance_to_anchor()</code> . . . . .	183
10.53.2.2 <code>get_optimal_partial_r_noised_vec()</code> . . . . .	183
10.53.2.3 <code>get_optimal_partial_ref_matrix()</code> . . . . .	184
10.53.2.4 <code>is_anchor()</code> . . . . .	184
10.53.2.5 <code>is_member()</code> . . . . .	185
10.53.2.6 <code>recog_mitigate_multipath()</code> . . . . .	185
10.53.2.7 <code>sim_UWB_dist()</code> . . . . .	187
10.54 <code>multipath_dist_detection_mitigation.h</code> File Reference . . . . .	188
10.54.1 Detailed Description . . . . .	188
10.54.2 Function Documentation . . . . .	188
10.54.2.1 <code>get_exact_distance_to_anchor()</code> . . . . .	188
10.54.2.2 <code>get_optimal_partial_r_noised_vec()</code> . . . . .	189
10.54.2.3 <code>get_optimal_partial_ref_matrix()</code> . . . . .	189
10.54.2.4 <code>is_anchor()</code> . . . . .	190
10.54.2.5 <code>is_member()</code> . . . . .	191
10.54.2.6 <code>recog_mitigate_multipath()</code> . . . . .	191
10.54.2.7 <code>sim_UWB_dist()</code> . . . . .	193
10.55 <code>newton_raphson.c</code> File Reference . . . . .	194
10.55.1 Detailed Description . . . . .	194
10.55.2 Function Documentation . . . . .	194

10.55.2.1 newton_raphson()	194
10.56 newton_raphson.h File Reference	195
10.56.1 Detailed Description	195
10.56.2 Function Documentation	196
10.56.2.1 newton_raphson()	196
10.57 norm_dist_rnd_generator.c File Reference	196
10.57.1 Detailed Description	197
10.57.2 Function Documentation	197
10.57.2.1 get_norm_distr_rand_num()	197
10.57.2.2 get_rand_num()	198
10.58 norm_dist_rnd_generator.h File Reference	198
10.58.1 Detailed Description	198
10.58.2 Macro Definition Documentation	199
10.58.2.1 PI	199
10.58.3 Function Documentation	199
10.58.3.1 get_norm_distr_rand_num()	199
10.58.3.2 get_rand_num()	199
10.59 optimization_test.c File Reference	200
10.59.1 Detailed Description	201
10.59.2 Function Documentation	201
10.59.2.1 optimization_exponential_data_test()	201
10.59.2.2 optimization_get_exp_f()	201
10.59.2.3 optimization_get_exp_Jacobian()	202
10.59.2.4 optimization_get_f_error()	202
10.59.2.5 optimization_get_J()	203
10.59.2.6 optimization_get_sin_f()	203
10.59.2.7 optimization_get_sin_Jacobian()	204
10.59.2.8 optimization_sinusoidal_data_test()	204
10.60 optimization_test.h File Reference	204
10.60.1 Detailed Description	205
10.60.2 Function Documentation	205
10.60.2.1 optimization_exponential_data_test()	205
10.60.2.2 optimization_sinusoidal_data_test()	205
10.61 pos_algos_common_test.c File Reference	206
10.61.1 Detailed Description	206
10.62 pos_algos_common_test.h File Reference	206
10.62.1 Detailed Description	206
10.63 position_optimization_test.c File Reference	207
10.63.1 Detailed Description	207
10.64 position_optimization_test.h File Reference	207
10.64.1 Detailed Description	207
10.65 pseudo_inverse.h File Reference	208

10.65.1 Detailed Description	208
10.65.2 Enumeration Type Documentation	208
10.65.2.1 ALGORITHM	208
10.66 qr_common.c File Reference	208
10.66.1 Detailed Description	209
10.66.2 Function Documentation	209
10.66.2.1 qr_common_backward_subst()	209
10.66.2.2 qr_common_get_reduced_QR()	210
10.67 qr_common.h File Reference	210
10.67.1 Detailed Description	211
10.67.2 Function Documentation	211
10.67.2.1 qr_common_backward_subst()	211
10.67.2.2 qr_common_get_reduced_QR()	211
10.68 qr_givens.c File Reference	212
10.68.1 Detailed Description	212
10.68.2 Function Documentation	212
10.68.2.1 qr_givens_decomp()	213
10.68.2.2 qr_givens_get_params()	213
10.69 qr_givens.h File Reference	214
10.69.1 Detailed Description	214
10.69.2 Function Documentation	214
10.69.2.1 qr_givens_decomp()	215
10.69.2.2 qr_givens_get_params()	215
10.70 qr_householder.c File Reference	216
10.70.1 Detailed Description	216
10.70.2 Function Documentation	216
10.70.2.1 qr_householder_decomp()	217
10.71 qr_householder.h File Reference	217
10.71.1 Detailed Description	218
10.71.2 Function Documentation	218
10.71.2.1 qr_householder_decomp()	218
10.72 qr_pinv_test.c File Reference	219
10.72.1 Detailed Description	219
10.73 qr_pinv_test.h File Reference	219
10.73.1 Detailed Description	219
10.74 qr_pseudo_inverse.c File Reference	220
10.74.1 Detailed Description	220
10.74.2 Function Documentation	220
10.74.2.1 qr_get_pinv()	220
10.75 qr_pseudo_inverse.h File Reference	221
10.75.1 Detailed Description	221
10.75.2 Function Documentation	221

10.75.2.1 qr_get_pinv()	221
10.76 shell_sort.c File Reference	222
10.76.1 Detailed Description	222
10.76.2 Function Documentation	222
10.76.2.1 int_shell_sort()	222
10.76.2.2 shell_sort()	223
10.77 shell_sort.h File Reference	223
10.77.1 Detailed Description	223
10.77.2 Function Documentation	224
10.77.2.1 int_shell_sort()	224
10.77.2.2 shell_sort()	224
10.78 solve.c File Reference	224
10.78.1 Detailed Description	225
10.78.2 Function Documentation	225
10.78.2.1 solve()	225
10.78.2.2 solve_givens()	226
10.78.2.3 solve_householder()	227
10.78.2.4 solve_lu_decomp()	227
10.79 solve.h File Reference	228
10.79.1 Detailed Description	228
10.79.2 Function Documentation	229
10.79.2.1 solve()	229
10.79.2.2 solve_givens()	229
10.79.2.3 solve_householder()	230
10.79.2.4 solve_lu_decomp()	231
10.80 solve_test.c File Reference	231
10.80.1 Detailed Description	232
10.81 solve_test.h File Reference	232
10.81.1 Detailed Description	232
10.82 svd.c File Reference	232
10.82.1 Detailed Description	233
10.82.2 Function Documentation	233
10.82.2.1 svd()	233
10.82.2.2 svd_compute_print_U_S_V_s()	234
10.82.2.3 svd_get_reciproc_singular_values()	235
10.82.2.4 svd_get_S_dim()	235
10.82.2.5 svd_get_single_values_num()	236
10.82.2.6 svd_get_U_dim()	236
10.82.2.7 svd_get_V_dim()	237
10.83 svd.h File Reference	237
10.83.1 Detailed Description	238
10.83.2 Function Documentation	238

10.83.2.1 svd()	238
10.83.2.2 svd_compute_print_U_S_V_s()	239
10.83.2.3 svd_get_reciproc_singular_values()	240
10.83.2.4 svd_get_S_dim()	240
10.83.2.5 svd_get_single_values_num()	241
10.83.2.6 svd_get_U_dim()	241
10.83.2.7 svd_get_V_dim()	242
10.84 svd_test.c File Reference	242
10.84.1 Detailed Description	242
10.85 svd_test.h File Reference	243
10.85.1 Detailed Description	243
10.86 trilateration.c File Reference	243
10.86.1 Detailed Description	244
10.86.2 Function Documentation	244
10.86.2.1 trilateration1()	244
10.86.2.2 trilateration2()	245
10.86.2.3 trilateration_get_A_matrix()	246
10.86.2.4 trilateration_get_b_vector()	246
10.86.2.5 trilateration_get_particular_solution()	247
10.86.2.6 trilateration_get_quadratic_equation_solution()	247
10.86.2.7 trilateration_get_rank_and_homogeneous_solution()	248
10.86.2.8 trilateration_preprocessed_get_particular_solution()	248
10.86.2.9 trilateration_solve_linear_equation()	249
10.87 trilateration.h File Reference	249
10.87.1 Detailed Description	250
10.87.2 Function Documentation	250
10.87.2.1 trilateration1()	251
10.87.2.2 trilateration2()	251
10.87.2.3 trilateration_get_A_matrix()	252
10.87.2.4 trilateration_get_b_vector()	253
10.87.2.5 trilateration_get_particular_solution()	253
10.87.2.6 trilateration_get_quadratic_equation_solution()	254
10.87.2.7 trilateration_get_rank_and_homogeneous_solution()	254
10.87.2.8 trilateration_preprocessed_get_particular_solution()	255
10.87.2.9 trilateration_solve_linear_equation()	255
10.88 utils.c File Reference	256
10.88.1 Detailed Description	257
10.88.2 Function Documentation	257
10.88.2.1 utils_get_median()	257
10.88.2.2 utils_get_save_square_root()	257
10.88.2.3 utils_max()	258
10.88.2.4 utils_mean()	258

10.88.2.5	<a href="#">utils_min()</a>	258
10.88.2.6	<a href="#">utils_moving_average()</a>	259
10.88.2.7	<a href="#">utils_printf()</a>	259
10.88.2.8	<a href="#">utils_sind()</a>	260
10.88.2.9	<a href="#">utils_swap()</a>	260
10.88.2.10	<a href="#">utils_to_radian()</a>	261
10.88.2.11	<a href="#">utils_u8_max()</a>	261
10.88.2.12	<a href="#">utils_u8_min()</a>	262
10.89	<a href="#">utils.h File Reference</a>	262
10.89.1	<a href="#">Detailed Description</a>	263
10.89.2	<a href="#">Function Documentation</a>	263
10.89.2.1	<a href="#">utils_get_median()</a>	263
10.89.2.2	<a href="#">utils_get_save_square_root()</a>	264
10.89.2.3	<a href="#">utils_max()</a>	264
10.89.2.4	<a href="#">utils_mean()</a>	265
10.89.2.5	<a href="#">utils_min()</a>	265
10.89.2.6	<a href="#">utils_moving_average()</a>	265
10.89.2.7	<a href="#">utils_printf()</a>	266
10.89.2.8	<a href="#">utils_sind()</a>	266
10.89.2.9	<a href="#">utils_swap()</a>	267
10.89.2.10	<a href="#">utils_to_radian()</a>	267
10.89.2.11	<a href="#">utils_u8_max()</a>	268
10.89.2.12	<a href="#">utils_u8_min()</a>	268
10.90	<a href="#">utils_test.c File Reference</a>	269
10.90.1	<a href="#">Detailed Description</a>	269
10.91	<a href="#">utils_test.h File Reference</a>	269
10.91.1	<a href="#">Detailed Description</a>	269
10.92	<a href="#">vector.c File Reference</a>	270
10.92.1	<a href="#">Detailed Description</a>	271
10.92.2	<a href="#">Function Documentation</a>	271
10.92.2.1	<a href="#">vector_add()</a>	271
10.92.2.2	<a href="#">vector_clear()</a>	272
10.92.2.3	<a href="#">vector_copy()</a>	272
10.92.2.4	<a href="#">vector_flex_print()</a>	273
10.92.2.5	<a href="#">vector_get_elements()</a>	273
10.92.2.6	<a href="#">vector_get_euclidean_distance()</a>	274
10.92.2.7	<a href="#">vector_get_index_vector()</a>	274
10.92.2.8	<a href="#">vector_get_max_and_index()</a>	275
10.92.2.9	<a href="#">vector_get_mean_value()</a>	275
10.92.2.10	<a href="#">vector_get_norm2()</a>	276
10.92.2.11	<a href="#">vector_get_residual()</a>	276
10.92.2.12	<a href="#">vector_get_scalar_product()</a>	277

10.92.2.13	<a href="#">vector_get_square_norm2()</a>	277
10.92.2.14	<a href="#">vector_get_sum()</a>	278
10.92.2.15	<a href="#">vector_in_place_scalar_mul()</a>	278
10.92.2.16	<a href="#">vector_is_equal()</a>	278
10.92.2.17	<a href="#">vector_mul()</a>	279
10.92.2.18	<a href="#">vector_print()</a>	279
10.92.2.19	<a href="#">vector_print_u8_array()</a>	280
10.92.2.20	<a href="#">vector_scalar_div()</a>	280
10.92.2.21	<a href="#">vector_scalar_mul()</a>	281
10.92.2.22	<a href="#">vector_square()</a>	281
10.92.2.23	<a href="#">vector_sub()</a>	281
10.92.2.24	<a href="#">vector_uint32_is_equal()</a>	282
10.93	<a href="#">vector.h File Reference</a>	282
10.93.1	<a href="#">Detailed Description</a>	284
10.93.2	<a href="#">Function Documentation</a>	284
10.93.2.1	<a href="#">vector_add()</a>	284
10.93.2.2	<a href="#">vector_clear()</a>	285
10.93.2.3	<a href="#">vector_copy()</a>	285
10.93.2.4	<a href="#">vector_flex_print()</a>	285
10.93.2.5	<a href="#">vector_get_elements()</a>	286
10.93.2.6	<a href="#">vector_get_euclidean_distance()</a>	286
10.93.2.7	<a href="#">vector_get_index_vector()</a>	287
10.93.2.8	<a href="#">vector_get_max_and_index()</a>	288
10.93.2.9	<a href="#">vector_get_mean_value()</a>	288
10.93.2.10	<a href="#">vector_get_norm2()</a>	289
10.93.2.11	<a href="#">vector_get_residual()</a>	289
10.93.2.12	<a href="#">vector_get_scalar_product()</a>	290
10.93.2.13	<a href="#">vector_get_square_norm2()</a>	290
10.93.2.14	<a href="#">vector_get_sum()</a>	291
10.93.2.15	<a href="#">vector_in_place_scalar_mul()</a>	291
10.93.2.16	<a href="#">vector_is_equal()</a>	291
10.93.2.17	<a href="#">vector_mul()</a>	292
10.93.2.18	<a href="#">vector_print()</a>	292
10.93.2.19	<a href="#">vector_print_u8_array()</a>	293
10.93.2.20	<a href="#">vector_scalar_div()</a>	293
10.93.2.21	<a href="#">vector_scalar_mul()</a>	294
10.93.2.22	<a href="#">vector_square()</a>	294
10.93.2.23	<a href="#">vector_sub()</a>	294
10.93.2.24	<a href="#">vector_uint32_is_equal()</a>	295
10.94	<a href="#">vector_test.c File Reference</a>	295
10.94.1	<a href="#">Detailed Description</a>	296
10.95	<a href="#">vector_test.h File Reference</a>	296

10.95.1 Detailed Description . . . . . 296



# Chapter 1

## RcdMathLib Documentation

Author

Zakaria Kasmi

### 1.1 RcdMathLib in a nutshell

RcdMathLib is an open-source library for numerical linear and non-linear algebra designed to match the requirements of resource-limited or embedded devices. RcdMathLib supports solving linear and non-linear equation systems. Furthermore, it provides general-purpose implemented methods that facilitate solving problems of regression smoothing and curve fitting. It also allows for calculating as well as optimizing a position on a mobile device.

### 1.2 Download and use the RcdMathLib

The simplest way to use the RcdMathLib for resource-limited devices is to download the [Eclipse project for resource-limited devices](#) zip file. Another [Eclipse project for full-fledged devices](#) zip file enables also the use of the RcdMathLib for full-fledged platforms or single-board computers such as a [Raspberry Pi](#).

### 1.3 The quickest start

You can run the RcdMathLib on most resource-limited devices such as the [STM32F4 Discovery board](#) as well as on full-fledged computers like a PC, or single-board devices such as a [Raspberry Pi](#). Try it right now in your terminal window:

```
mkdir RcdMathLib_resource_limited_devices
cd RcdMathLib_resource_limited_devices
git clone https://git.imp.fu-berlin.de/zkasmi/RcdMathLib.git # assumption: git is pre-installed
unzip RcdMathLib_resource_limited_devices.zip

cd RcdMathLib/examples/linear_algebra/basic_operation/
make all
```

The example above shows how to use the basic operations of the RcdMathLib such as the vector or the matrix operations.

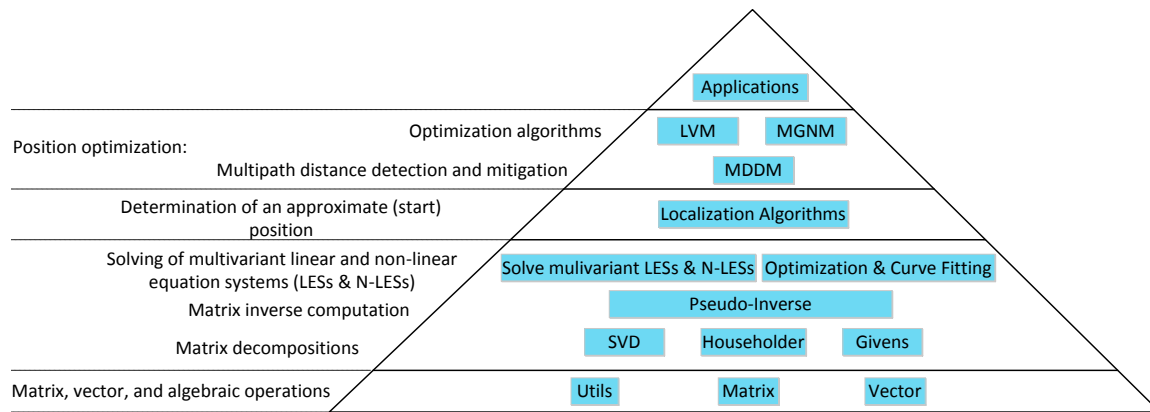


Figure 1.1 Example of real-world localization system

## 1.4 Structure

This section walks you through RcdMathLib's structure to easily find your way around in RcdMathLib's code base.

RcdMathLib's source code is composed into three groups:

- Linear algebra
- Non-linear algebra
- Localization

In addition RcdMathLib includes various [examples](#) to familiarize the user with the software as well as an API to facilitate the use and the further development of the library. The structural groups are projected onto the [directory structure](#) of RcdMathLib, where each of these groups resides in one or two directories in the [main](#) RcdMathLib directory.

The following list gives a more detailed description of each of RcdMathLib's top-level directories:

### 1.4.1 Linear Algebra

This directory contains functions that are specific to vector and matrix operations, and other algebraic operations. It provides functions to perform basic matrix operations such as matrix addition, multiplication, or transposition. It also provides algorithms for complex operations like matrix decomposition algorithms, algorithms to calculate the pseudo-inverse of a matrix, or methods to solve systems of linear equations. The linear algebra module is divided in the following sub-modules:

- Basic operations sub-module.
- Matrix decompositions sub-module.
- Pseudo-Inverse sub-module.
- Solve linear equations sub-module.
- Utilities sub-module.

See the [LINEAR\\_ALGEBRA](#) module in the [main](#) RcdMathLib directory as well as in the [documentation](#) repository directory for further information and API documentations.

### 1.4.2 Non-Linear Algebra

The non-linear algebra module contains functions to solve multi-variant nonlinear equations as well as algorithms solving problems of regression smoothing and curve fitting. This module also enables the optimization of an approximate solution by using Non-linear Least Squares (NLS) methods such as modified Gauss–Newton (GN) or the Levenberg–Marquardt (LVM) algorithms. The non-linear algebra module is divided into the two following sub-modules:

- Solve non-linear equations sub-module.
- Optimization sub-module.

See here [NON\\_LINEAR\\_ALGEBRA](#) module for further information.

### 1.4.3 Localization

The localization module contains functions to compute a position of a mobile device using distance measurements or DC-pulsed, magnetic signals. This module also includes optimization algorithms such as the Levenberg–Marquardt approach to optimize the calculated position. The localization module also involves a method to recognize and mitigate the multipath errors on the mobile station.

In the `position_algos` sub-directory you can find the implementations of the distance-based localization system (see the [DISTANCE\\_BASED](#) module) as well as the implementations of the DC-pulsed, magnetic position system (see the [MAGNETIC\\_BASED](#) module). The `pos_algos_common` sub-directory contains common localization algorithms like the trilateration method (see the [POS\\_ALGOS\\_COMMON](#) module). The optimization algorithms are located in the `optimization` sub-directory providing the following optimization approaches:

- Gauss–Newton (GN) algorithm.
- Levenberg–Marquardt (LVM) algorithm.
- Multipath Distance Detection and Mitigation (MDDM) algorithm.

See the [LOCALIZATION](#) module for more detailed information.

### 1.4.4 examples

Here you find a number of example applications that demonstrate certain features of RcdMathLib. The examples found in this directory is a good starting point for anyone who is new to RcdMathLib.

For more information best browse that directory and have a look at the `README.md` files that ship with each example.

To create your own application - here or anywhere else - see [Creating an application](#)

### 1.4.5 doc

The `doc` directory contains the doxygen configuration and also contains the compiled doxygen output after running `make doc`.

## 1.5 Further information

- [Getting started](#)
- [Creating an application](#)
- [Creating modules](#)



## Chapter 2

# Getting started

Author

Zakaria Kasmi

## 2.1 Downloading RcdMathLib code

You can obtain the latest RcdMathLib code from our [GitLab](#) repository either by [downloading the Eclipse project for resource-limited devices](#) zip file, the [Eclipse project for full-fledged devices](#) zip file, or by cloning the [git repository](#).

In order to clone the RcdMathLib repository, you need the [Git revision control system](#) and run the following command:

```
git clone https://git.imp.fu-berlin.de/zkasmircdmathlib.git
```

## 2.2 Compiling RcdMathLib

### 2.2.1 Setting up a toolchain for full-fledged devices

RcdMathLib can be run on full-fledged devices such as a Personal Computer (PC) or a server. It can also be executed on the embedded system Raspberry Pi running an Operating System (OS) such as the Raspbian or the Ubuntu Core. We recommend the GNU Compiler Collection (GCC) to compile the RcdMathLib on operating systems like the Linux or Windows. We also recommend the use of the [Eclipse \(IDE\)](#).

The PATH environment variable should be set with bin-directory of the GCC compiler. On a typical shell like bash or zsh this can be done using export, e.g.

```
export PATH=${PATH}:/path/to/gcc/bin
```

For windows, add the bin-directory to your PATH under System Properties->Advanced-> Environment Variables, e.g.

```
"C:\gcc\bin"
```

## 2.2.2 Setting up a toolchain for resource-limited devices

Depending on the hardware you want to use, you need to first install a corresponding toolchain. We recommend the use of the [RIOT-OS](#), since it supports 8-bit platforms (e.g. Arduino Mega 2560), 16-bit platforms (e.g. MSP430), and 32-bit platforms (e.g. ARM). The [RIOT-OS](#) is an open source IoT operating system developed at the "Freie Universität Berlin". In general, an OS allows the management and sharing of resources as well as the development of multi-tasking applications in a computer system. The [Wiki](#) on RIOT's Github page contains a lot of information that can help you with your platform:

- [ARM-based platforms](#)
- [TI MSP430](#)
- [Atmel ATmega](#)
- [native](#)

## 2.2.3 The build system for full-fledged devices

RcdMathLib can be built using the [Eclipse IDE](#) for C/C++ Developers. The simplest way to compile and link an application with the RcdMathLib is: Firstly, download the [RcdMathLib\\_full\\_fledged\\_devices.zip](#) file or clone it as follows:

```
git clone https://git.imp.fu-berlin.de/zkasmi/RcdMathLib.git
cd RcdMathLib/eclipse_projects/
```

After cloning or downloading the RcdMathLib, the Eclipse version of the RcdMathLib can be used as follows:

- Create a workspace folder (e.g. `rcd_math_lib_workspace`).
- Change the current directory to the "`rcd_math_lib_workspace`" directory and unzip the `RcdMathLib_full_fledged_devices.zip` file.  

```
cd rcd_math_lib_workspace
unzip RcdMathLib_full_fledged_devices.zip
```
- Start the Eclipse IDE and browse to the created workspace folder.
- Compile the whole project.

Another method to use the RcdMathLib in the Eclipse IDE is:

- Clone the whole:  

```
git clone https://git.imp.fu-berlin.de/zkasmi/RcdMathLib.git
```
- Create a new C project by choosing the project type under Executable: Hello World ANSI C Project, as well as the Cross GCC Tool-chains.
- Try to compile and execute the Hello World ANSI C Project.
- Copy the RcdMathLib as well as the [eclipse\\_path\\_includes\\_settings.xml](#) configuration file into the src-directory of your project.
- Import the 'eclipse\_path\_includes\_settings.xml' file as follows:
  - Open File -> Import -> C/C++ and Select "C/C++ Project Settings" from the Selection Wizard.
  - Click Next->"Select Settings file"-->Browse.
  - After Browsing to the xml-configuration file, click on Finish.
- Compile the whole project.
- Optional: replace the `main()`-function with the `main.c` file of the cloned RcdMathLib version for full-fledged devices.

### 2.2.4 The build system for resource-limited devices

RcdMathLib and RIOT use [GNU make](#) as build system. The simplest way to compile and link an application with RcdMathLib, is to use the [Eclipse project for resource-limited devices](#) and set up a Makefile providing at least the following variables:

- `APPLICATION`: should contain the (unique) name of your application
- `BOARD`: specifies the platform the application should be build for by default
- `RIOTBASE`: specifies the path to the copy of the RIOT repository (note, the `macro` can be used to give a relative path)
- `RCDMATHLIB`: specifies the path to the copy of the RcdMathLib repository (note, the `macro` can be used to give a relative path)
- `USEMODULE`: specifies the module of the RcdMathLib that you may want to use

Additionally it has to include the `Makefile.include`, located in RcdMathLib's as well as RIOT's root directories.

A minimal application Makefile looks like this:

```
# a minimal application Makefile
APPLICATION = mini-makefile
BOARD ?= native
RIOTBASE ?= $(CURDIR)/../RIOT
RCDMATHLIB ?= $(CURDIR)/../RcdMathLib
USEMODULE += basic_operations
include $(RIOTBASE)/MicroPosMath-Lib/Makefile.include
include $(RIOTBASE)/Makefile.include
```

The `?=` operator can be used in order to allow overwriting variables from the command line. For example, the target platform can be specified from the command line as follows:

```
make BOARD=stm32f4discovery
```

In this case the STM32F4 discovery board is specified. Furthermore, the basic operations sub-module is selected by the `USEMODULE` macro.

Other sub-modules such as the matrix decompositions or utilities modules can be selected, e.g.:

```
USEMODULE += matrix_decompositions
USEMODULE += utilities
```

The dependency of the RcdMathLib modules and sub-modules to each other is automatically calculated by special Makefiles located in the module directories.

Besides typical targets like `clean`, `all`, or `doc`, RIOT provides the special targets `flash` and `term` to invoke the configured flashing and terminal tools for the specified platform. These targets use the variable `PORT` for the serial communication to the device. Neither this variable nor the targets `flash` and `term` are mandatory for the native port.

Unless specified otherwise, make will create an elf-file as well as an Intel hex file in the `bin` folder of your application directory.

Please visit the [Wiki](#) to learn more about the build system of the RIOT.

## 2.2.5 Building and executing an example for resource-limited devices

RcdMathLib provides a number of examples in the [examples-directory](#). Every example has a README that documents its usage and its purpose. Furthermore the examples are described in the [doc-directory](#). You can build your own application or the examples by typing:

```
make BOARD=stm32f4discovery
```

or

```
make all BOARD=stm32f4discovery
```

into your shell.

To flash the application to a board just type

```
make flash BOARD=stm32f4discovery
```

You can then access the board via the serial interface:

```
make term BOARD=stm32f4discovery
```

If you are using multiple boards you can use the `PORT` macro to specify the serial interface:

```
make term BOARD=stm32f4discovery PORT=/dev/ttyACM1
```

We use `pyterm` as the default terminal application. It is shipped with RIOT in the `dist/tools/pyterm/` directory. If you choose to use another terminal program you can set `TERMPROG` (and if need be the `TERMFLAGS`) macros:

```
make -C examples/gnrc_networking/ term \  
    BOARD=samr21-xpro \  
    TERMPROG=gtkterm \  
    TERMFLAGS="-s 115200 -p /dev/ttyACM0 -e"
```

Please visit the [Wiki](#) to learn more about flashing devices.

## Chapter 3

# Creating an application

Author

Zakaria Kasmi

An application can be created for full-fledged or resource-limited devices.

### 3.1 Creating an application for full-fledged devices

We recommend to use the [Eclipse IDE](#) for C/C++ Developers for creating own application. The simplest way to write your own application, is to put your \*.c and \*.h files under the `src` directory. Another way is to create a directory containing the multiple C file(s) with your source code. The header files can be imported in the Eclipse IDE by opening Project->Properties and selecting "C/C++ General->Paths and Symbols->Languages->GNU C" from the Selection Wizard. Click the "Add.." button to browse to the header-directory and the select the "Apply" and "OK" buttons. The users can be oriented to the `main.c` and the examples in the [main-directory](#) of the RcdMathLib.

### 3.2 Creating an application for resource-limited devices

To create your own application for a resource-limited device you need to create a directory containing one or multiple C file(s) with your source code and a Makefile. An example Makefile is available in the `src` folder of the [Eclipse project for resource-limited devices](#).

### 3.3 The main function

RIOT starts two threads the idle and main threads after the board is initialized. The idle thread has the lowest priority while the main thread has a priority that is in the middle between the lowest and the highest available priorities. The main thread is the first that runs and calls the `main()` function. This function needs to be defined in the source code of each application (typically located in the `main.c` file).

```
#include <stdio.h>
#include "matrix_test.h"
#include "vector_test.h"
int main(void)
{
    puts("RcdMathLib Application!");
    // Test the basic operations module;
    matrix_test();
    vector_test();

    return EXIT_SUCCESS;
}
```

The above C code shows an application testing the basic operations sub-modules. This application run operations of the vector and matrix sub-modules.

## 3.4 The application's Makefile

### 3.4.1 The minimal Makefile

At minimum the Makefile of an application (see [Getting started](#)) needs to define the following macros:

- `APPLICATION`: contains the name of your application
- `RIOTBASE`: specifies the path to your copy of the RIOT repository (note, the macro can be used to give a relative path)
- `RCDMATHLIB`: specifies the path to the copy of the RcdMathLib repository (note, the macro can be used to give a relative path)
- `USEMODULE`: specifies the module of the RcdMathLib that you may want to use

The `BOARD` macro is also required and recommended to be set to `native` by default, but is recommended to be overridable with the `?=` operator. Additionally, it is required to include the `Makefile.include` from the RcdMathLib and from the RIOTBASE.

```
# Set the name of your application:
APPLICATION = foobar
# If no BOARD is found in the environment, use this default:
BOARD ?= native
# This has to be the absolute path to the RcdMathLib base directory:
RCDMATHLIB ?= $(CURDIR)/../RcdMathLib
# This has to be the absolute path to the RIOT base directory:
RIOTBASE ?= $(CURDIR)/../../RIOT
include $(RCDMATHLIB)/Makefile.include
include $(RIOTBASE)/Makefile.include
```

### 3.4.2 Including modules

The modules of the RcdMathLib as well as of the RIOT can be included. In order to use additional modules, such as a particular driver or a system library, the modules' names must be appended to the `USEMODULE` variable. For example, to build an application using the SHT11 temperature sensor and UDP/IPV6 functionalities, the Makefile needs to contain the following lines:

```
USEMODULE += sht11
USEMODULE += gnrc_ipv6_default
USEMODULE += gnrc_udp
```

For example, to create an application using the matrix decompositions, the pseudo-inverse, and solving linear equations sub-modules, the Makefile must comprise the following lines:

```
USEMODULE += matrix_decompositions
USEMODULE += pseudo_inverse
USEMODULE += utilities
```

## Chapter 4

# Creating modules

Author

Zakaria Kasmi

Well-defined units of code in the RcdMathLib that provide a set of features are encapsulated in a module. RcdMathLib is module-based and composed of the following main modules:

- Linear algebra module
- Non-Linear algebra module
- Localization module

Each main module includes sub-modules, for more details see [The structure of the RcdMathLib](#).

Note

The following chapters concerning only resource-limited devices.

### 4.1 The general structure

Modules are directories containing source and header files as well as a Makefile. Furthermore, their API can be defined in one or more header files, residing in the include path of their super-module.

For example, the matrix sub-module is implemented in the [BASIC\\_OPERATIONS](#) sub-module, in the `linear_algebra/basic_operations` directory. Its API is defined in `linear_algebra/basic_operations/matrix.h`.

A module's Makefile just needs to include `Makefile.base` in the RIOT repository as well as `Makefile.base` and `Makefile.include` in the RcdMathLib repository:

```
include $(RIOTBASE)/Makefile.base
include $(RCDMATHLIB)/linear_algebra/matrix_decompositions/Makefile.include
include $(RCDMATHLIB)/linear_algebra/matrix_decompositions/Makefile.dep
```

The `Makefile.base` and `Makefile.include` macros in the example above are includes for the linear algebra module.

If your module's name differs from the name of the directory it resides in you need to set the `MODULE` macro in addition.

The `Makefile.dep` serves to define dependencies and the `Makefile.include` to append target specific information to variables like `INCLUDES`. Modules can be used by adding their name to the `USEMODULE` macro of the application's Makefile.

## 4.2 Module dependencies

The module may depend on other modules to minimize code duplication. These dependencies are defined in `Makefile.dep` with the following syntax:

```
ifneq (, $(filter your_module, $(USEMODULE))) # if module in USEMODULE
    USEMODULE += dep1                        # add dependencies to USEMODULE
    USEMODULE += dep2
endif
```

### Note

`Makefile.dep` is processed only once therefore, the dependency block for a module must be added *before* dependencies pull in their dependencies.

## Chapter 5

# Module Index

### 5.1 Modules

Here is a list of all modules:

EXAMPLES . . . . .	23
LINEAR_ALGEBRA . . . . .	24
BASIC_OPERATIONS . . . . .	21
MATRIX_DECOMPOSITIONS . . . . .	27
PSEUDO_INVERSE . . . . .	33
SOLVE_LINEAR_EQUATIONS . . . . .	34
UTILITIES . . . . .	36
LOCALIZATION . . . . .	25
POSITION_ALGOS . . . . .	30
DISTANCE_BASED . . . . .	22
MAGNETIC_BASED . . . . .	26
POS_ALGOS_COMMON . . . . .	32
POSITION_OPTIMIZATION . . . . .	31
NON_LINEAR_ALGEBRA . . . . .	28
OPTIMIZATION . . . . .	29
SOLVE_NON_LINEAR_EQUATIONS . . . . .	35



## Chapter 6

# Data Structure Index

### 6.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">matrix_dim_t</a>	A structure to define the row and column number of a matrix . . . . .	<a href="#">37</a>
------------------------------	---	--------------------



## Chapter 7

# File Index

### 7.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">combinatorics.c</a>	Calculate possible $\binom{n}{k}$ combinations without repetition in ascending order . . . . .	39
<a href="#">combinatorics.h</a>	Calculate possible $\binom{n}{k}$ combinations without repetition in ascending order . . . . .	40
<a href="#">damped_newton_raphson.c</a>	Implement the damped Newton–Raphson algorithm . . . . .	42
<a href="#">damped_newton_raphson.h</a>	Implement the damped Newton–Raphson algorithm . . . . .	45
<a href="#">dist_based_fi.c</a>	Error function of distance-based localization systems . . . . .	48
<a href="#">dist_based_fi.h</a>	Error function of distance-based localization systems . . . . .	50
<a href="#">dist_based_jacobian.c</a>	Jacobian function of distance-based localization systems . . . . .	51
<a href="#">dist_based_jacobian.h</a>	Jacobian function of distance-based localization systems . . . . .	54
<a href="#">dist_based_position.c</a>	Functions of distance-based localization systems . . . . .	57
<a href="#">dist_based_position.h</a>	Functions of distance-based localization systems . . . . .	59
<a href="#">distance_based_test.c</a>	Examples of localization algorithms of distance-based positioning systems . . . . .	60
<a href="#">distance_based_test.h</a>	Examples of localization algorithms of distance-based positioning systems . . . . .	61
<a href="#">DOP.c</a>	Compute the Position Dilution of Precision (PDOP) . . . . .	62
<a href="#">DOP.h</a>	Compute the Position Dilution of Precision (PDOP) . . . . .	63
<a href="#">fsolve.c</a>	Solve multi-variant nonlinear equation systems . . . . .	64
<a href="#">fsolve.h</a>	Solve multi-variant nonlinear equation systems . . . . .	66
<a href="#">fsolve_test.c</a>	Examples of solving non-linear equation systems . . . . .	68
<a href="#">fsolve_test.h</a>	Examples of solving non-linear equation systems . . . . .	72

<a href="#">givens_test.c</a>	Examples of the Givens algorithm . . . . .	72
<a href="#">givens_test.h</a>	Examples of the Givens algorithm . . . . .	73
<a href="#">householder_test.c</a>	Examples of the Householder algorithm . . . . .	73
<a href="#">householder_test.h</a>	Examples of the Householder algorithm . . . . .	74
<a href="#">levenberg_marquardt.c</a>	Implement the Levenberg–Marquardt (LVM) algorithm . . . . .	74
<a href="#">levenberg_marquardt.h</a>	Implement the Levenberg–Marquardt (LVM) algorithm . . . . .	78
<a href="#">loc_gauss_newton.c</a>	Implement the Gauss–Newton algorithm . . . . .	80
<a href="#">loc_gauss_newton.h</a>	Implement the Gauss–Newton algorithm for position optimization . . . . .	82
<a href="#">loc_levenberg_marquardt.c</a>	Implement the Levenberg–Marquardt (LVM) algorithm . . . . .	83
<a href="#">loc_levenberg_marquardt.h</a>	Implement the Levenberg–Marquardt (LVM) algorithm for position optimization . . . . .	88
<a href="#">lu_decomp.c</a>	Computes the LU decomposition of the matrix . . . . .	93
<a href="#">lu_decomp.h</a>	Computes the LU decomposition of the matrix . . . . .	94
<a href="#">lu_decomp_test.c</a>	Examples of the LU algorithm with pivoting . . . . .	96
<a href="#">lu_decomp_test.h</a>	Examples of the LU algorithm with pivoting . . . . .	96
<a href="#">magnetic_based_fi.c</a>	Error function of DC-pulsed, magnetic localization system . . . . .	97
<a href="#">magnetic_based_fi.h</a>	Error function of DC-pulsed, magnetic localization system . . . . .	98
<a href="#">magnetic_based_jacobian.c</a>	Jacobian function of DC-pulsed, magnetic localization system . . . . .	99
<a href="#">magnetic_based_jacobian.h</a>	Jacobian function of DC-pulsed, magnetic localization system . . . . .	102
<a href="#">magnetic_based_position.c</a>	Functions of DC-pulsed, magnetic localization system . . . . .	105
<a href="#">magnetic_based_position.h</a>	Functions of DC-pulsed, magnetic localization system . . . . .	110
<a href="#">magnetic_based_test.c</a>	Examples of localization algorithms of magnetic-based positioning systems . . . . .	115
<a href="#">magnetic_based_test.h</a>	Examples of localization algorithms of magnetic-based positioning systems . . . . .	115
<a href="#">matrix.c</a>	Matrix computations . . . . .	116
<a href="#">matrix.h</a>	Matrix computations . . . . .	143
<a href="#">matrix_test.c</a>	Examples of matrix computations . . . . .	171
<a href="#">matrix_test.h</a>	Examples of matrix computations . . . . .	172
<a href="#">modified_gauss_newton.c</a>	Implement the Gauss–Newton algorithm . . . . .	172
<a href="#">modified_gauss_newton.h</a>	Implement the Gauss–Newton algorithm . . . . .	174
<a href="#">moore_penrose_pinv_test.c</a>	Examples of the Moore–Penrose algorithm . . . . .	176

<a href="#">moore_penrose_pinv_test.h</a>	Examples of the Moore–Penrose algorithm . . . . .	176
<a href="#">moore_penrose_pseudo_inverse.c</a>	Moore–Penrose algorithm to compute the pseudo-inverse of a rectangular matrix . . . . .	177
<a href="#">moore_penrose_pseudo_inverse.h</a>	Moore–Penrose algorithm to compute the pseudo-inverse of a matrix . . . . .	178
<a href="#">multipath_algo_own_norm_distr_test.c</a>	Example of the algorithm for the recognition and mitigation of multipath effects . . . . .	180
<a href="#">multipath_algo_own_norm_distr_test.h</a>	Example of the algorithm for the recognition and mitigation of multipath effects . . . . .	181
<a href="#">multipath_dist_detection_mitigation.c</a>	Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm . . . . .	182
<a href="#">multipath_dist_detection_mitigation.h</a>	Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm . . . . .	188
<a href="#">newton_raphson.c</a>	Implement the Newton–Raphson algorithm . . . . .	194
<a href="#">newton_raphson.h</a>	Implement the Newton–Raphson algorithm . . . . .	195
<a href="#">norm_dist_rnd_generator.c</a>	Generating normally distributed random numbers . . . . .	196
<a href="#">norm_dist_rnd_generator.h</a>	Generating normally distributed random numbers . . . . .	198
<a href="#">optimization_test.c</a>	Examples of optimization algorithms . . . . .	200
<a href="#">optimization_test.h</a>	Examples of optimization algorithms . . . . .	204
<a href="#">pos_algos_common_test.c</a>	Examples of common algorithms of localization systems . . . . .	206
<a href="#">pos_algos_common_test.h</a>	Examples of common algorithms of localization systems . . . . .	206
<a href="#">position_optimization_test.c</a>	Examples of optimization algorithms for localization systems . . . . .	207
<a href="#">position_optimization_test.h</a>	Examples of optimization algorithms for localization systems . . . . .	207
<a href="#">pseudo_inverse.h</a>	Compute the pseudo-inverse of a matrix . . . . .	208
<a href="#">qr_common.c</a>	Common definitions and implementations for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using . . . . .	208
<a href="#">qr_common.h</a>	Common definitions and implementations for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using . . . . .	210
<a href="#">qr_givens.c</a>	Givens algorithm for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using Givens rotations. $A = QR$ , where Q is an $(m \times n)$ -matrix with orthonormal columns and R is an $(n \times n)$ upper triangular matrix . . . . .	212
<a href="#">qr_givens.h</a>	Givens algorithm for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using Givens rotations. $A = QR$ , where Q is an $(m \times n)$ -matrix with orthonormal columns and R is an $(n \times n)$ upper triangular matrix . . . . .	214
<a href="#">qr_householder.c</a>	Householder algorithm for the QR-decomposition . . . . .	216
<a href="#">qr_householder.h</a>	Householder algorithm for the QR-decomposition . . . . .	217
<a href="#">qr_pinv_test.c</a>	Examples of the QR-based pseudo-inverse algorithm . . . . .	219
<a href="#">qr_pinv_test.h</a>	Examples of the QR-based pseudo-inverse algorithm . . . . .	219

<a href="#">qr_pseudo_inverse.c</a>	QR decomposition algorithms to compute the pseudo-inverse of a matrix . . . . .	220
<a href="#">qr_pseudo_inverse.h</a>	QR decomposition algorithms to compute the pseudo-inverse of a matrix . . . . .	221
<a href="#">shell_sort.c</a>	Implement the Shell sort algorithm . . . . .	222
<a href="#">shell_sort.h</a>	Implement the Shell sort algorithm . . . . .	223
<a href="#">solve.c</a>	Enables to solve systems of linear equations $Ax = b$ for $x$ . . . . .	224
<a href="#">solve.h</a>	Enables to solve systems of linear equations $Ax = b$ for $x$ . . . . .	228
<a href="#">solve_test.c</a>	Examples of solving linear equation systems . . . . .	231
<a href="#">solve_test.h</a>	Examples of solving linear equation systems . . . . .	232
<a href="#">svd.c</a>	Algorithm for the Singular Value Decomposition (SVD). Provide necessary methods to compute the compact SVD of a matrix. $A = U \cdot S \cdot V$ , where $U$ is a $(m \times l)$ orthogonal matrix, $S$ is a $(l \times l)$ diagonal matrix, $V$ is a $(l \times n)$ orthogonal matrix, and $l = \min(m, n)$ . The SVD is computed by using the Golub–Kahan–Reinsch algorithm that works in two phases: bidiagonalization and a reduction to the diagonal form phase . . . . .	232
<a href="#">svd.h</a>	Algorithm for the Singular Value Decomposition (SVD) . . . . .	237
<a href="#">svd_test.c</a>	Examples of the SVD algorithm . . . . .	242
<a href="#">svd_test.h</a>	Examples of the SVD algorithm . . . . .	243
<a href="#">trilateration.c</a>	Implement the trilateration algorithm . . . . .	243
<a href="#">trilateration.h</a>	Implement the trilateration algorithm . . . . .	249
<a href="#">utils.c</a>	Utilities for linear algebra. Utility-functions are needed by the linear algebra-module as well as other modules such as the position algorithm-module . . . . .	256
<a href="#">utils.h</a>	Utilities for linear algebra . . . . .	262
<a href="#">utils_test.c</a>	Examples of the utility functions . . . . .	269
<a href="#">utils_test.h</a>	Examples of the utility functions . . . . .	269
<a href="#">vector.c</a>	Vector computations. Vector computations include operations such as addition, subtraction, and inner product (dot product) . . . . .	270
<a href="#">vector.h</a>	Vector computations . . . . .	282
<a href="#">vector_test.c</a>	Examples of vector computations . . . . .	295
<a href="#">vector_test.h</a>	Examples of vector computations . . . . .	296

## Chapter 8

# Module Documentation

### 8.1 BASIC\_OPERATIONS

Matrix basic operations.

#### Files

- file [matrix.h](#)  
*Matrix computations.*
- file [vector.h](#)  
*Vector computations.*

#### 8.1.1 Detailed Description

Matrix basic operations.

This module provides functions to perform basic matrix operations such as matrix addition, multiplication, or transposition.

#### Author

Zakaria Kasmi

## 8.2 DISTANCE\_BASED

localization algorithms of distance-based localization systems.

### Files

- file [dist\\_based.fi.h](#)  
*Error function of distance-based localization systems.*
- file [dist\\_based\\_jacobian.h](#)  
*Jacobian function of distance-based localization systems.*
- file [dist\\_based\\_position.h](#)  
*Functions of distance-based localization systems.*

### 8.2.1 Detailed Description

localization algorithms of distance-based localization systems.

The localization module contains functions to compute a position of a mobile device using distance measurements signals.

#### Author

Zakaria Kasmi

## 8.3 EXAMPLES

Examples of the RcdMathLib.

### Files

- file [matrix\\_test.h](#)  
*Examples of matrix computations.*
- file [vector\\_test.h](#)  
*Examples of vector computations.*
- file [givens\\_test.h](#)  
*Examples of the Givens algorithm.*
- file [householder\\_test.h](#)  
*Examples of the Householder algorithm.*
- file [lu\\_decomp\\_test.h](#)  
*Examples of the LU algorithm with pivoting.*
- file [svd\\_test.h](#)  
*Examples of the SVD algorithm.*
- file [moore\\_penrose\\_pinv\\_test.h](#)  
*Examples of the Moore–Penrose algorithm.*
- file [qr\\_pinv\\_test.h](#)  
*Examples of the QR-based pseudo-inverse algorithm.*
- file [solve\\_test.h](#)  
*Examples of solving linear equation systems.*
- file [utils\\_test.h](#)  
*Examples of the utility functions.*
- file [distance\\_based\\_test.h](#)  
*Examples of localization algorithms of distance-based positioning systems.*
- file [magnetic\\_based\\_test.h](#)  
*Examples of localization algorithms of magnetic-based positioning systems.*
- file [pos\\_algos\\_common\\_test.h](#)  
*Examples of common algorithms of localization systems.*
- file [multipath\\_algo\\_own\\_norm\\_distr\\_test.h](#)  
*Example of the algorithm for the recognition and mitigation of multipath effects.*
- file [position\\_optimization\\_test.h](#)  
*Examples of optimization algorithms for localization systems.*
- file [optimization\\_test.h](#)  
*Examples of optimization algorithms.*
- file [fsolve\\_test.h](#)  
*Examples of solving non-linear equation systems.*

### 8.3.1 Detailed Description

Examples of the RcdMathLib.

Includes examples of all RcdMathLib's modules.

Author

Zakaria Kasmi

## 8.4 LINEAR\_ALGEBRA

Linear algebra operations.

### Modules

- [BASIC\\_OPERATIONS](#)  
*Matrix basic operations.*
- [MATRIX\\_DECOMPOSITIONS](#)  
*Matrix decomposition algorithms.*
- [PSEUDO\\_INVERSE](#)  
*Algorithms to calculate the pseudo-inverse of a matrix.*
- [SOLVE\\_LINEAR\\_EQUATIONS](#)  
*Enables to solve systems of linear equations  $Ax = b$  for  $x$ .*
- [UTILITIES](#)  
*Utilities for linear algebra.*

### 8.4.1 Detailed Description

Linear algebra operations.

The linear algebra module contains functions that are specific to vector and matrix operations, and other algebraic operations. This module is composed of five submodules: `basic_operation`, `matrix_decompositions`, `pseudo_inverse`, `solve_linear_equations`, and `utilities` submodules.

#### Author

Zakaria Kasmi

## 8.5 LOCALIZATION

localization and optimization algorithms of distance- and magnetic-based localization systems.

### Modules

- [POSITION\\_ALGOS](#)  
*localization algorithms of distance- and magnetic-based localization systems.*
- [POSITION\\_OPTIMIZATION](#)  
*position optimization of distance- and magnetic-based localization systems.*

### 8.5.1 Detailed Description

localization and optimization algorithms of distance- and magnetic-based localization systems.

The localization module contains functions to compute a position of a mobile device using distance measurements or DC-pulsed, magnetic signals. This module also includes optimization algorithms such as the Levenberg–Marquardt approach to optimize the calculated position. The localization module also involves a method to recognize and mitigate the multipath errors on the mobile station.

#### Author

Zakaria Kasmi

## 8.6 MAGNETIC\_BASED

localization algorithms of magnetic-based localization systems.

### Files

- file [magnetic\\_based\\_fi.h](#)  
*Error function of DC-pulsed, magnetic localization system.*
- file [magnetic\\_based\\_jacobian.h](#)  
*Jacobian function of DC-pulsed, magnetic localization system.*
- file [magnetic\\_based\\_position.h](#)  
*Functions of of DC-pulsed, magnetic localization system.*

### 8.6.1 Detailed Description

localization algorithms of magnetic-based localization systems.

The localization module contains functions to compute a position of a mobile device using artificially generated DC-pulsed, magnetic signals.

#### Author

Zakaria Kasmi

## 8.7 MATRIX\_DECOMPOSITIONS

Matrix decomposition algorithms.

### Files

- file [lu\\_decomp.h](#)  
*Computes the LU decomposition of the matrix.*
- file [qr\\_common.h](#)  
*Common definitions and implementations for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using.*
- file [qr\\_givens.h](#)  
*Givens algorithm for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using Givens rotations.  $A = QR$ , where  $Q$  is an  $(m \times n)$ -matrix with orthonormal columns and  $R$  is an  $(n \times n)$  upper triangular matrix.*
- file [qr\\_householder.h](#)  
*Householder algorithm for the QR-decomposition.*
- file [svd.h](#)  
*Algorithm for the Singular Value Decomposition (SVD).*

### 8.7.1 Detailed Description

Matrix decomposition algorithms.

This module provides algorithms of matrix decomposition such as Householder, Givens, or Golub-Kahan-Reinsch algorithms.

#### Author

Zakaria Kasmi

## 8.8 NON\_LINEAR\_ALGEBRA

Non-linear algebra operations.

### Modules

- [OPTIMIZATION](#)  
*Solving problems of regression smoothing and curve fitting.*
- [SOLVE\\_NON\\_LINEAR\\_EQUATIONS](#)  
*Enables to solve multi-variant nonlinear equation systems.*

### 8.8.1 Detailed Description

Non-linear algebra operations.

The non-linear algebra module contains functions to solve multi-variant nonlinear equations as wells algorithms solving problems of regression smoothing and curve fitting. This module also enables enables the optimization of an approximate solution by using Non-linear Least Squares (NLS) methods such as modified Gauss–Newton (GN) or the Levenberg–Marquardt (LVM) algorithms.

#### Author

Zakaria Kasmi

## 8.9 OPTIMIZATION

Solving problems of regression smoothing and curve fitting.

### Files

- file [levenberg\\_marquardt.h](#)  
*Implement the Levenberg–Marquardt (LVM) algorithm.*
- file [modified\\_gauss\\_newton.h](#)  
*Implement the Gauss–Newton algorithm.*

### 8.9.1 Detailed Description

Solving problems of regression smoothing and curve fitting.

This module also enables the optimization of an approximate solution by using Non-linear Least Squares (NLS) methods such as modified Gauss–Newton (GN) or the Levenberg–Marquardt (LVM) algorithms.

#### Author

Zakaria Kasmi

## 8.10 POSITION\_ALGOS

localization algorithms of distance- and magnetic-based localization systems.

### Modules

- [DISTANCE\\_BASED](#)  
*localization algorithms of distance-based localization systems.*
- [MAGNETIC\\_BASED](#)  
*localization algorithms of magnetic-based localization systems.*
- [POS\\_ALGOS\\_COMMON](#)  
*Common algorithms of localization systems.*

### 8.10.1 Detailed Description

localization algorithms of distance- and magnetic-based localization systems.

The localization module contains functions to compute a position of a mobile device using distance measurements or DC-pulsed, magnetic signals.

#### Author

Zakaria Kasmi

## 8.11 POSITION\_OPTIMIZATION

position optimization of distance- and magnetic-based localization systems.

### Files

- file [loc\\_gauss\\_newton.h](#)  
*Implement the Gauss–Newton algorithm for position optimization.*
- file [loc\\_levenberg\\_marquardt.h](#)  
*Implement the Levenberg–Marquardt (LVM) algorithm for position optimization.*
- file [multipath\\_dist\\_detection\\_mitigation.h](#)  
*Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm.*

### 8.11.1 Detailed Description

position optimization of distance- and magnetic-based localization systems.

The optimization module enables the refinement of a position of a mobile device using algorithms such as Gauss–Newton, Levenberg, or a Multipath Distance Detection and Mitigation (MDDM) algorithm.

#### Author

Zakaria Kasmi

## 8.12 POS\_ALGOS\_COMMON

Common algorithms of localization systems.

### Files

- file [DOP.h](#)  
*Compute the Position Dilution of Precision (PDOP).*
- file [trilateration.h](#)  
*Implement the trilateration algorithm.*

### 8.12.1 Detailed Description

Common algorithms of localization systems.

This module contains common functions such as the trilateration or the Position Dilution of Precision (PDOP).

#### Author

Zakaria Kasmi

## 8.13 PSEUDO\_INVERSE

Algorithms to calculate the pseudo-inverse of a matrix.

### Files

- file [moore\\_penrose\\_pseudo\\_inverse.h](#)  
*Moore–Penrose algorithm to compute the pseudo-inverse of a matrix.*
- file [pseudo\\_inverse.h](#)  
*Compute the pseudo-inverse of a matrix.*
- file [qr\\_pseudo\\_inverse.h](#)  
*QR decomposition algorithms to compute the pseudo-inverse of a matrix.*

### 8.13.1 Detailed Description

Algorithms to calculate the pseudo-inverse of a matrix.

The pseudo-inverse matrix can be computed using QR-decomposition or SVD algorithms.

#### Author

Zakaria Kasmi

## 8.14 SOLVE\_LINEAR\_EQUATIONS

Enables to solve systems of linear equations  $Ax = b$  for  $x$ .

### Files

- file [solve.h](#)

*Enables to solve systems of linear equations  $Ax = b$  for  $x$ .*

### 8.14.1 Detailed Description

Enables to solve systems of linear equations  $Ax = b$  for  $x$ .

The user can select various algorithm such as the Moore–Penrose inverse, the Givens or the Householder algorithm for the QR-decomposition to solve the systems of linear equations.

#### Author

Zakaria Kasmi

## 8.15 SOLVE\_NON\_LINEAR\_EQUATIONS

Enables to solve multi-variant nonlinear equation systems.

### Files

- file [damped\\_newton\\_raphson.h](#)  
*Implement the damped Newton–Raphson algorithm.*
- file [fsolve.h](#)  
*Solve multi-variant nonlinear equation systems.*
- file [newton\\_raphson.h](#)  
*Implement the Newton–Raphson algorithm.*

### 8.15.1 Detailed Description

Enables to solve multi-variant nonlinear equation systems.

The multi-variant nonlinear equation systems are solved using damped or the Newton–Raphson algorithms.

#### Author

Zakaria Kasmi

## 8.16 UTILITIES

Utilities for linear algebra.

### Files

- file [combinatorics.h](#)  
*Calculate possible  $\binom{n}{k}$  combinations without repetition in ascending order.*
- file [norm\\_dist\\_rnd\\_generator.h](#)  
*Generating normally distributed random numbers.*
- file [shell\\_sort.h](#)  
*Implement the Shell sort algorithm.*
- file [utils.h](#)  
*Utilities for linear algebra.*

### 8.16.1 Detailed Description

Utilities for linear algebra.

Utility-functions are needed by linear algebra-module as well as other modules such as the position algorithm-module.

#### Author

Zakaria Kasmi

## Chapter 9

# Data Structure Documentation

### 9.1 `matrix_dim_t` Struct Reference

A structure to define the row and column number of a matrix.

```
#include <matrix.h>
```

#### Data Fields

- `uint8_t row_num`  
*the row number*
- `uint8_t col_num`  
*the column number*

#### 9.1.1 Detailed Description

A structure to define the row and column number of a matrix.

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

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

- `matrix.h`



## Chapter 10

# File Documentation

### 10.1 combinatorics.c File Reference

Calculate possible  $\binom{n}{k}$  combinations without repetition in ascending order.

```
#include <stdio.h>
#include <stdlib.h>
#include "combinatorics.h"
#include "vector.h"
```

#### Functions

- `uint8_t combinatorics_init (uint8_t n, uint8_t k, uint8_t comb_arr[])`  
*Initialize the combinations generator.*
- `uint8_t combinatorics_get_next_without_rep (uint8_t n, uint8_t k, uint8_t comb_arr[])`  
*Generate the next combination.*

#### 10.1.1 Detailed Description

Calculate possible  $\binom{n}{k}$  combinations without repetition in ascending order.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

#### 10.1.2 Function Documentation

##### 10.1.2.1 combinatorics\_get\_next\_without\_rep()

```
uint8_t combinatorics_get_next_without_rep (
    uint8_t n,
    uint8_t k,
    uint8_t comb_arr[] )
```

Generate the next combination.

**Parameters**

in	$n$	size of the set.
in	$k$	size of the sub-set.
in, out	<i>comb_arr[]</i>	pointer to the combination set.

return [COMBI\\_END](#), if the last combination is generated. return [COMBI\\_SUCCESS](#), if successful.

Definition at line 48 of file combinatorics.c.

References [COMBI\\_END](#), and [COMBI\\_SUCCESS](#).

Referenced by [recog\\_mitigate\\_multipath\(\)](#).

**10.1.2.2 combinatorics\_init()**

```
uint8_t combinatorics_init (
    uint8_t n,
    uint8_t k,
    uint8_t comb_arr[] )
```

Initialize the combinations generator.

**Parameters**

in	$n$	size of the set.
in	$k$	size of the sub-set.
out	<i>comb_arr[]</i>	pointer to the combination set.

return [COMBI\\_ERROR](#), if  $k > n$ . return [COMBI\\_EMPTY](#), if  $k = 0$ . return [COMBI\\_SUCCESS](#), if successful.

Definition at line 29 of file combinatorics.c.

References [COMBI\\_EMPTY](#), [COMBI\\_ERROR](#), and [COMBI\\_SUCCESS](#).

Referenced by [recog\\_mitigate\\_multipath\(\)](#).

**10.2 combinatorics.h File Reference**

Calculate possible  $\binom{n}{k}$  combinations without repetition in ascending order.

```
#include <stdint.h>
```

## Macros

- #define `COMBI_ERROR` -1  
*Case of an error.*
- #define `COMBI_EMPTY` 0  
*Case of an empty combination set.*
- #define `COMBI_SUCCESS` 1  
*Case of successfully calculated combination set.*
- #define `COMBI_END` 2  
*Case of completion of calculating combination sets.*

## Functions

- `uint8_t combinatorics_init` (`uint8_t n`, `uint8_t k`, `uint8_t comb_arr[]`)  
*Initialize the combinations generator.*
- `uint8_t combinatorics_get_next_without_rep` (`uint8_t n`, `uint8_t k`, `uint8_t comb_arr[]`)  
*Generate the next combination.*

### 10.2.1 Detailed Description

Calculate possible  $\binom{n}{k}$  combinations without repetition in ascending order.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.2.2 Function Documentation

#### 10.2.2.1 `combinatorics_get_next_without_rep()`

```
uint8_t combinatorics_get_next_without_rep (
    uint8_t n,
    uint8_t k,
    uint8_t comb_arr[] )
```

Generate the next combination.

#### Parameters

<code>in</code>	<code>n</code>	size of the set.
<code>in</code>	<code>k</code>	size of the sub-set.
<code>in, out</code>	<code>comb_arr[]</code>	pointer to the combination set.

return `COMBI_END`, if the last combination is generated. return `COMBI_SUCCESS`, if successful.

Definition at line 48 of file `combinatorics.c`.

References COMBI\_END, and COMBI\_SUCCESS.

Referenced by recog\_mitigate\_multipath().

### 10.2.2.2 combinatorics\_init()

```
uint8_t combinatorics_init (
    uint8_t n,
    uint8_t k,
    uint8_t comb_arr[] )
```

Initialize the combinations generator.

#### Parameters

in	$n$	size of the set.
in	$k$	size of the sub-set.
out	<code>comb_arr[]</code>	pointer to the combination set.

return COMBI\_ERROR, if  $k > n$ . return COMBI\_EMPTY, if  $k = 0$ . return COMBI\_SUCCESS, if successful.

Definition at line 29 of file combinatorics.c.

References COMBI\_EMPTY, COMBI\_ERROR, and COMBI\_SUCCESS.

Referenced by recog\_mitigate\_multipath().

## 10.3 damped\_newton\_raphson.c File Reference

Implement the damped Newton–Raphson algorithm.

```
#include <stdio.h>
#include "moore_penrose_pseudo_inverse.h"
#include "damped_newton_raphson.h"
#include "vector.h"
#include "matrix.h"
```

### Functions

- `uint8_t damped_newton_raphson` (`uint8_t f_length`, `uint8_t n`, `vector_t x0_arr[]`, `double min_lamda`, `double eps`, `uint8_t max_it_num`, `vector_t est_x_arr[]`, `void(*get_non_lin_sys)(vector_t x_arr[], vector_t f_vec[])`, `void(*get_jacobian)(vector_t x_arr[], matrix_t J[][n])`)  
*Implements the damped Newton–Raphson algorithm.*
- `double get_damped_norm` (`uint8_t m`, `uint8_t n`, `vector_t x_arr[]`, `void(*get_non_lin_sys)(vector_t x_arr[], vector_t f_vec[])`, `void(*get_jacobian)(vector_t x_arr[], matrix_t J[][n])`)  
*Compute the norm of the damped Newton–Raphson algorithm.*
- `void get_delta_x` (`uint8_t m`, `uint8_t n`, `vector_t x_arr[]`, `void(*get_non_lin_sys)(vector_t x_arr[], vector_t f_vec[])`, `void(*get_jacobian)(vector_t x_arr[], matrix_t J[][n])`, `vector_t delta_x_arr[]`)  
*Compute the correction vector the damped Newton–Raphson algorithm.*

### 10.3.1 Detailed Description

Implement the damped Newton–Raphson algorithm.

The damped Newton–Raphson algorithm enables to solve multi-variant nonlinear equation systems.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.3.2 Function Documentation

#### 10.3.2.1 damped\_newton\_raphson()

```
uint8_t damped_newton_raphson (
    uint8_t f_length,
    uint8_t n,
    vector_t x0_arr[],
    double min_lamda,
    double eps,
    uint8_t max_it_num,
    vector_t est_x_arr[],
    void(*) (vector_t x_arr[], vector_t f_vec[]) get_non_lin_sys,
    void(*) (vector_t x_arr[], matrix_t J[][n]) get_jacobian )
```

Implements the damped Newton–Raphson algorithm.

The user should provide pointers to non-linear equation systems and Jacobian functions.

Note

This function is generally implemented.

Parameters

in	<i>f_length</i>	length of the error functions vector.
in	<i>n</i>	length of the start vector.
in	<i>x0_arr[]</i>	start vector.
in	<i>min_lamda</i>	minimal damping factor.
in	<i>eps</i>	accuracy bound.
in	<i>max_it_num</i>	maximal iteration number of the damped Newton–Raphson algorithm.
out	<i>est_x_arr[]</i>	estimated (solution) vector.
in	<i>(*get_non_lin_sys)</i>	pointer to non-linear equation systems.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

**Returns**

required iteration number.

Definition at line 32 of file damped\_newton\_raphson.c.

References `get_damped_norm()`, `get_delta_x()`, `vector_add()`, `vector_copy()`, `vector_get_norm2()`, `vector_scalar_↵_mul()`, and `vector_t`.

Referenced by `fsolve()`.

**10.3.2.2 get\_damped\_norm()**

```
double get_damped_norm (
    uint8_t m,
    uint8_t n,
    vector_t x_arr[],
    void(*) (vector_t x_arr[], vector_t f_vec[]) get_non_lin_sys,
    void(*) (vector_t x_arr[], matrix_t J[][n]) get_jacobian )
```

Compute the norm of the damped Newton–Raphson algorithm.

The user should provide pointers to non-linear equation systems and Jacobian functions.

**Note**

This function is generally implemented.

**Parameters**

in	<i>m</i>	number of the non-linear equations.
in	<i>n</i>	length of the guess vector.
in	<i>x_arr[]</i>	guess vector.
in	<i>(*get_non_lin_sys)</i>	pointer to non-linear equation systems.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

**Returns**

norm of the damped Newton–Raphson algorithm.

Definition at line 104 of file damped\_newton\_raphson.c.

References `get_delta_x()`, `vector_get_norm2()`, and `vector_t`.

Referenced by `damped_newton_raphson()`.

### 10.3.2.3 get\_delta\_x()

```
void get_delta_x (
    uint8_t m,
    uint8_t n,
    vector_t x_arr[],
    void(*) (vector_t x_arr[], vector_t f_vec[]) get_non_lin_sys,
    void(*) (vector_t x_arr[], matrix_t J[][n]) get_jacobian,
    vector_t delta_x_arr[] )
```

Compute the correction vector the damped Newton–Raphson algorithm.

The user should provide pointers to non-linear equation systems and Jacobian functions.

#### Note

This function is generally implemented.

#### Parameters

in	<i>m</i>	number of the non-linear equations.
in	<i>n</i>	length of the guess vector.
in	<i>x_arr[]</i>	guess vector.
in	<i>(*get_non_lin_sys)</i>	pointer to non-linear equation systems.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.
in, out	<i>delta_x_arr[]</i>	the correction vector (term).

Definition at line 120 of file damped\_newton\_raphson.c.

References `matrix_mul_vec()`, `matrix_t`, `moore_penrose_get_pinv()`, `vector_in_place_scalar_mul()`, and `vector_t`.

Referenced by `damped_newton_raphson()`, and `get_damped_norm()`.

## 10.4 damped\_newton\_raphson.h File Reference

Implement the damped Newton–Raphson algorithm.

```
#include "matrix.h"
#include "vector.h"
```

### Functions

- `uint8_t damped_newton_raphson` (`uint8_t f_length`, `uint8_t n`, `vector_t x0_arr[]`, `double min_lambda`, `double eps`, `uint8_t max_it_num`, `vector_t est_x_arr[]`, `void(*get_non_lin_sys)(vector_t x_arr[], vector_t f_vec[])`, `void(*get_jacobian)(vector_t x_arr[], matrix_t J[][n])`)  
*Implements the damped Newton–Raphson algorithm.*
- `double get_damped_norm` (`uint8_t m`, `uint8_t n`, `vector_t x_arr[]`, `void(*get_non_lin_sys)(vector_t x_arr[], vector_t f_vec[])`, `void(*get_jacobian)(vector_t x_arr[], matrix_t J[][n])`)  
*Compute the norm of the damped Newton–Raphson algorithm.*
- `void get_delta_x` (`uint8_t m`, `uint8_t n`, `vector_t x_arr[]`, `void(*get_non_lin_sys)(vector_t x_arr[], vector_t f_vec[])`, `void(*get_jacobian)(vector_t x_arr[], matrix_t J[][n])`, `vector_t delta_x_arr[]`)  
*Compute the correction vector the damped Newton–Raphson algorithm.*

### 10.4.1 Detailed Description

Implement the damped Newton–Raphson algorithm.

The damped Newton–Raphson algorithm enables to solve multi-variant nonlinear equation systems.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.4.2 Function Documentation

#### 10.4.2.1 damped\_newton\_raphson()

```
uint8_t damped_newton_raphson (
    uint8_t f_length,
    uint8_t n,
    vector_t x0_arr[],
    double min_lamda,
    double eps,
    uint8_t max_it_num,
    vector_t est_x_arr[],
    void(*) (vector_t x_arr[], vector_t f_vec[]) get_non_lin_sys,
    void(*) (vector_t x_arr[], matrix_t J[][n]) get_jacobian )
```

Implements the damped Newton–Raphson algorithm.

The user should provide pointers to non-linear equation systems and Jacobian functions.

Note

This function is generally implemented.

Parameters

in	<i>f_length</i>	length of the error functions vector.
in	<i>n</i>	length of the start vector.
in	<i>x0_arr[]</i>	start vector.
in	<i>min_lamda</i>	minimal damping factor.
in	<i>eps</i>	accuracy bound.
in	<i>max_it_num</i>	maximal iteration number of the damped Newton–Raphson algorithm.
out	<i>est_x_arr[]</i>	estimated (solution) vector.
in	<i>(*get_non_lin_sys)</i>	pointer to non-linear equation systems.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

**Returns**

required iteration number.

Definition at line 32 of file damped\_newton\_raphson.c.

References `get_damped_norm()`, `get_delta_x()`, `vector_add()`, `vector_copy()`, `vector_get_norm2()`, `vector_scalar_↵_mul()`, and `vector_t`.

Referenced by `fsolve()`.

**10.4.2.2 get\_damped\_norm()**

```
double get_damped_norm (
    uint8_t m,
    uint8_t n,
    vector_t x_arr[],
    void(*) (vector_t x_arr[], vector_t f_vec[]) get_non_lin_sys,
    void(*) (vector_t x_arr[], matrix_t J[][n]) get_jacobian )
```

Compute the norm of the damped Newton–Raphson algorithm.

The user should provide pointers to non-linear equation systems and Jacobian functions.

**Note**

This function is generally implemented.

**Parameters**

in	<i>m</i>	number of the non-linear equations.
in	<i>n</i>	length of the guess vector.
in	<i>x_arr[]</i>	guess vector.
in	<i>(*get_non_lin_sys)</i>	pointer to non-linear equation systems.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

**Returns**

norm of the damped Newton–Raphson algorithm.

Definition at line 104 of file damped\_newton\_raphson.c.

References `get_delta_x()`, `vector_get_norm2()`, and `vector_t`.

Referenced by `damped_newton_raphson()`.

### 10.4.2.3 get\_delta\_x()

```
void get_delta_x (
    uint8_t m,
    uint8_t n,
    vector_t x_arr[],
    void(*) (vector_t x_arr[], vector_t f_vec[]) get_non_lin_sys,
    void(*) (vector_t x_arr[], matrix_t J[][n]) get_jacobian,
    vector_t delta_x_arr[] )
```

Compute the correction vector the damped Newton–Raphson algorithm.

The user should provide pointers to non-linear equation systems and Jacobian functions.

#### Note

This function is generally implemented.

#### Parameters

in	<i>m</i>	number of the non-linear equations.
in	<i>n</i>	length of the guess vector.
in	<i>x_arr[]</i>	guess vector.
in	<i>(*get_non_lin_sys)</i>	pointer to non-linear equation systems.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.
in, out	<i>delta_x_arr[]</i>	the correction vector (term).

Definition at line 120 of file damped\_newton\_raphson.c.

References `matrix_mul_vec()`, `matrix_t`, `moore_penrose_get_pinv()`, `vector_in_place_scalar_mul()`, and `vector_t`.

Referenced by `damped_newton_raphson()`, and `get_damped_norm()`.

## 10.5 dist\_based\_fi.c File Reference

Error function of distance-based localization systems.

```
#include <math.h>
#include "matrix.h"
```

### Functions

- `matrix_t dist_based_fi (matrix_t point[3], matrix_t ref_point[3], matrix_t ri)`  
*Defines the error function of a distance-based localization system.*
- void `dist_based_f_i (uint8_t ref_points_num, matrix_t ref_point_mat[ref_points_num][3], matrix_t point[3], matrix_t d_vec[], matrix_t f_vec[])`  
*Defines the error function of a distance-based localization system.*

## 10.5.1 Detailed Description

Error function of distance-based localization systems.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.5.2 Function Documentation

### 10.5.2.1 dist\_based\_f\_i()

```
void dist_based_f_i (
    uint8_t ref_points_num,
    matrix_t ref_point_mat[ref_points_num][3],
    matrix_t point[3],
    matrix_t d_vec[],
    matrix_t f_vec[] )
```

Defines the error function of a distance-based localization system.

This error function is related to multiple reference stations.

Parameters

in	<i>ref_points_num</i>	Number of the reference stations.
in	<i>ref_point_mat</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>d_vec</i> []	distances to the reference stations.
out	<i>f_vec</i> []	errors related to reference stations and destined position.

Definition at line 47 of file dist\_based\_fi.c.

Referenced by `multipath_algo_own_norm_distr_test()`, and `position_optimization_test()`.

### 10.5.2.2 dist\_based\_fi()

```
matrix_t dist_based_fi (
    matrix_t point[3],
    matrix_t ref_point[3],
    matrix_t ri )
```

Defines the error function of a distance-based localization system.

This error function is related to one reference station.

**Parameters**

in	<i>point[]</i>	three-dimensional coordinates of the mobile device.
in	<i>ref_point[]</i>	three-dimensional coordinates of a reference station.
in	<i>ri</i>	distance to a reference station.

**Returns**

the error related to a reference station and destined position.

Definition at line 26 of file `dist_based_fi.c`.

References `matrix_t`.

Referenced by `dist_based_jacobian_get_JTf()`, and `dist_based_jacobian_get_JTJ()`.

## 10.6 `dist_based_fi.h` File Reference

Error function of distance-based localization systems.

```
#include "matrix.h"
```

**Functions**

- `matrix_t dist_based_fi (matrix_t point[3], matrix_t ref_point[3], matrix_t ri)`  
*Defines the error function of a distance-based localization system.*
- void `dist_based_f_i (uint8_t ref_points_num, matrix_t ref_point_mat[ref_points_num][3], matrix_t point[3], matrix_t d_vec[], matrix_t f_vec[])`  
*Defines the error function of a distance-based localization system.*

### 10.6.1 Detailed Description

Error function of distance-based localization systems.

**Author**

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.6.2 Function Documentation

#### 10.6.2.1 `dist_based_f_i()`

```
void dist_based_f_i (
    uint8_t ref_points_num,
    matrix_t ref_point_mat[ref_points_num][3],
    matrix_t point[3],
    matrix_t d_vec[],
    matrix_t f_vec[] )
```

Defines the error function of a distance-based localization system.

This error function is related to multiple reference stations.

## Parameters

in	<i>ref_points_num</i>	Number of the reference stations.
in	<i>ref_point_mat</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>d_vec</i> []	distances to the reference stations.
out	<i>f_vec</i> []	errors related to reference stations and destined position.

Definition at line 47 of file dist\_based\_fi.c.

Referenced by multipath\_algo\_own\_norm\_distr\_test(), and position\_optimization\_test().

## 10.6.2.2 dist\_based\_fi()

```
matrix_t dist_based_fi (
    matrix_t point[3],
    matrix_t ref_point[3],
    matrix_t ri )
```

Defines the error function of a distance-based localization system.

This error function is related to one reference station.

## Parameters

in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>ref_point</i> []	three-dimensional coordinates of a reference station.
in	<i>ri</i>	distance to a reference station.

## Returns

the error related to a reference station and destined position.

Definition at line 26 of file dist\_based\_fi.c.

References matrix\_t.

Referenced by dist\_based\_jacobian\_get\_JTf(), and dist\_based\_jacobian\_get\_JTJ().

## 10.7 dist\_based\_jacobian.c File Reference

Jacobian function of distance-based localization systems.

```
#include <math.h>
#include "matrix.h"
#include "vector.h"
#include "dist_based_fi.h"
```

## Functions

- void `dist_based_jacobian_get_JTf` (uint8\_t ref\_points\_num, matrix\_t ref\_point\_matrix[ref\_points\_num][3], matrix\_t point[3], matrix\_t dist\_vec[ref\_points\_num], vector\_t JTf[3])  
*Defines  $J_f^T \vec{f}$  of distance-based localization system.*
- void `dist_based_jacobian_get_JTJ` (uint8\_t ref\_points\_num, matrix\_t ref\_point\_matrix[ref\_points\_num][3], matrix\_t point[3], matrix\_t dist\_vec[ref\_points\_num], matrix\_t JTJ[3][3])  
*Defines  $J_f^T J_f$  of distance-based localization system.*
- void `dist_based_jacobian_get_J` (uint8\_t ref\_points\_num, matrix\_t point[3], matrix\_t ref\_point\_matrix[ref\_points\_num][3], matrix\_t J[ref\_points\_num][3])  
*Computes the Jacobian matrix of distance-based localization system.*
- void `dist_based_jacobian_get_J_mul_s` (uint8\_t ref\_points\_num, matrix\_t ref\_point\_matrix[ref\_points\_num][3], matrix\_t point[3], matrix\_t s[3], matrix\_t J\_s[ref\_points\_num])  
*Computes  $J_f^T \vec{s}$  of distance-based localization system.*

### 10.7.1 Detailed Description

Jacobian function of distance-based localization systems.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.7.2 Function Documentation

#### 10.7.2.1 dist\_based\_jacobian\_get\_J()

```
void dist_based_jacobian_get_J (
    uint8_t ref_points_num,
    matrix_t point[3],
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t J[ref_points_num][3] )
```

Computes the Jacobian matrix of distance-based localization system.

#### Parameters

in	<code>ref_points_num</code>	number of the reference stations.
in	<code>point[]</code>	three-dimensional coordinates of the mobile device.
in	<code>ref_point_matrix[][]</code>	three-dimensional coordinates of the reference stations.
out	<code>J[][]</code>	includes the Jacobian Matrix.

Definition at line 100 of file `dist_based_jacobian.c`.

References `matrix_t`.

### 10.7.2.2 dist\_based\_jacobian\_get\_J\_mul\_s()

```
void dist_based_jacobian_get_J_mul_s (
    uint8_t ref_points_num,
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t s[3],
    matrix_t J_s[ref_points_num] )
```

Computes  $J_f^T \vec{s}$  of distance-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>s</i> []	correction vector.
out	<i>J_s</i> []	includes the $J_f^T \vec{s}$ vector.

Definition at line 132 of file dist\_based\_jacobian.c.

References matrix\_t.

Referenced by multipath\_algo\_own\_norm\_distr\_test(), and position\_optimization\_test().

### 10.7.2.3 dist\_based\_jacobian\_get\_JTf()

```
void dist_based_jacobian_get_JTf (
    uint8_t ref_points_num,
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t dist_vec[ref_points_num],
    vector_t JTf[3] )
```

Defines  $J_f^T \vec{f}$  of distance-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>dist_vec</i> []	distances to the reference stations.
in, out	<i>JTf</i> []	includes the $J_f^T \vec{f}$ vector.

Definition at line 28 of file dist\_based\_jacobian.c.

References `dist_based_fi()`, `matrix_t`, and `vector_clear()`.

Referenced by `multipath_algo_own_norm_distr_test()`, and `position_optimization_test()`.

#### 10.7.2.4 `dist_based_jacobian_get_JTJ()`

```
void dist_based_jacobian_get_JTJ (
    uint8_t ref_points_num,
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t dist_vec[ref_points_num],
    matrix_t JTJ[3][3] )
```

Defines  $J_f^T J_f$  of distance-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

##### Parameters

in	<code>ref_points_num</code>	number of the reference stations.
in	<code>ref_point_matrix[ ][ ]</code>	three-dimensional coordinates of the reference stations.
in	<code>point[ ]</code>	three-dimensional coordinates of the mobile device.
in	<code>dist_vec[ ][ ]</code>	distances to the reference stations.
out	<code>JTJ[ ][ ]</code>	includes the $J_f^T J_f$ matrix.

Definition at line 56 of file `dist_based_jacobian.c`.

References `dist_based_fi()`, `matrix_clear()`, and `matrix_t`.

Referenced by `multipath_algo_own_norm_distr_test()`, and `position_optimization_test()`.

## 10.8 `dist_based_jacobian.h` File Reference

Jacobian function of distance-based localization systems.

```
#include "matrix.h"
#include "vector.h"
```

### Functions

- void `dist_based_jacobian_get_JTf` (uint8\_t ref\_points\_num, matrix\_t ref\_point\_matrix[ref\_points\_num][3], matrix\_t point[3], matrix\_t dist\_vec[ref\_points\_num], vector\_t JTf[3])  
Defines  $J_f^T \vec{f}$  of distance-based localization system.
- void `dist_based_jacobian_get_JTJ` (uint8\_t ref\_points\_num, matrix\_t ref\_point\_matrix[ref\_points\_num][3], matrix\_t point[3], matrix\_t dist\_vec[ref\_points\_num], matrix\_t JTJ[3][3])  
Defines  $J_f^T J_f$  of distance-based localization system.

- void `dist_based_jacobian_get_J_mul_s` (uint8\_t ref\_points\_num, matrix\_t ref\_point\_matrix[ref\_points\_num][3], matrix\_t point[3], matrix\_t s[3], matrix\_t J\_s[ref\_points\_num])  
*Computes  $J_f^T \vec{s}$  of distance-based localization system.*
- void `dist_based_jacobian_get_J` (uint8\_t ref\_points\_num, matrix\_t point[3], matrix\_t ref\_point\_matrix[ref\_points\_num][3], matrix\_t J[ref\_points\_num][3])  
*Computes the Jacobian matrix of distance-based localization system.*

### 10.8.1 Detailed Description

Jacobian function of distance-based localization systems.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.8.2 Function Documentation

#### 10.8.2.1 dist\_based\_jacobian\_get\_J()

```
void dist_based_jacobian_get_J (
    uint8_t ref_points_num,
    matrix_t point[3],
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t J[ref_points_num][3] )
```

Computes the Jacobian matrix of distance-based localization system.

Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>point[]</i>	three-dimensional coordinates of the mobile device.
in	<i>ref_point_matrix[][]</i>	three-dimensional coordinates of the reference stations.
out	<i>J[][]</i>	includes the Jacobian Matrix.

Definition at line 100 of file `dist_based_jacobian.c`.

References `matrix_t`.

#### 10.8.2.2 dist\_based\_jacobian\_get\_J\_mul\_s()

```
void dist_based_jacobian_get_J_mul_s (
    uint8_t ref_points_num,
    matrix_t ref_point_matrix[ref_points_num][3],
```

```

matrix_t point[3],
matrix_t s[3],
matrix_t J_s[ref_points_num] )

```

Computes  $J_f^T \vec{s}$  of distance-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [[[]]]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>s</i> []	correction vector.
out	<i>J_s</i> []	includes the $J_f^T \vec{s}$ vector.

Definition at line 132 of file `dist_based_jacobian.c`.

References `matrix_t`.

Referenced by `multipath_algo_own_norm_distr_test()`, and `position_optimization_test()`.

#### 10.8.2.3 dist\_based\_jacobian\_get\_JTf()

```

void dist_based_jacobian_get_JTf (
    uint8_t ref_points_num,
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t dist_vec[ref_points_num],
    vector_t JTf[3] )

```

Defines  $J_f^T \vec{f}$  of distance-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [[[]]]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>dist_vec</i> []	distances to the reference stations.
in, out	<i>JTf</i> []	includes the $J_f^T \vec{f}$ vector.

Definition at line 28 of file `dist_based_jacobian.c`.

References `dist_based_fi()`, `matrix_t`, and `vector_clear()`.

Referenced by `multipath_algo_own_norm_distr_test()`, and `position_optimization_test()`.

### 10.8.2.4 dist\_based\_jacobian\_get\_JTJ()

```
void dist_based_jacobian_get_JTJ (
    uint8_t ref_points_num,
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t dist_vec[ref_points_num],
    matrix_t JTJ[3][3] )
```

Defines  $J_f^T J_f$  of distance-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>dist_vec</i> [][]	distances to the reference stations.
out	<i>JTJ</i> [][]	includes the $J_f^T J_f$ matrix.

Definition at line 56 of file dist\_based\_jacobian.c.

References dist\_based\_fi(), matrix\_clear(), and matrix\_t.

Referenced by multipath\_algo\_own\_norm\_distr\_test(), and position\_optimization\_test().

## 10.9 dist\_based\_position.c File Reference

Functions of distance-based localization systems.

```
#include "dist_based_position.h"
#include <math.h>
#include "matrix.h"
#include "vector.h"
```

### Functions

- void [dist\\_based\\_get\\_absolute\\_error](#) (matrix\_t value\_arr[], matrix\_t approx\_value\_arr[], matrix\_t absolute\_error\_arr[], uint8\_t length)  
*Computes the absolute error of a position of a distance-based localization system.*
- matrix\_t [dist\\_based\\_get\\_distance\\_to\\_anchor](#) (matrix\_t ref\_point[3], matrix\_t point[3])  
*Computes the distance between a mobile station and a reference station of a distance-based localization system.*

### 10.9.1 Detailed Description

Functions of distance-based localization systems.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.9.2 Function Documentation

### 10.9.2.1 `dist_based_get_absolute_error()`

```
void dist_based_get_absolute_error (
    matrix_t value_arr[],
    matrix_t approx_value_arr[],
    matrix_t absolute_error_arr[],
    uint8_t length )
```

Computes the absolute error of a position of a distance-based localization system.

#### Parameters

in	<i>value_arr[]</i>	true position.
in	<i>approx_value_arr[]</i>	approximate position of the mobile device.
in, out	<i>absolute_error_arr[]</i>	includes the absolute error.
in	<i>length</i>	arrays length.

Definition at line 29 of file `dist_based_position.c`.

### 10.9.2.2 `dist_based_get_distance_to_anchor()`

```
matrix_t dist_based_get_distance_to_anchor (
    matrix_t ref_point[3],
    matrix_t point[3] )
```

Computes the distance between a mobile station and a reference station of a distance-based localization system.

#### Parameters

in	<i>ref_point</i>	three-dimensional coordinates of the reference stations.
in	<i>point</i>	three-dimensional coordinates of the mobile device.

#### Returns

the distance between the mobile station and the reference station.

Definition at line 45 of file `dist_based_position.c`.

References `matrix_t`, `vector_get_norm2()`, and `vector_sub()`.

## 10.10 dist\_based\_position.h File Reference

Functions of distance-based localization systems.

```
#include <math.h>
#include <inttypes.h>
#include "matrix.h"
```

### Functions

- void [dist\\_based\\_get\\_absolute\\_error](#) ([matrix\\_t](#) value\_arr[], [matrix\\_t](#) approx\_value\_arr[], [matrix\\_t](#) absolute\_error\_arr[], [uint8\\_t](#) length)  
*Computes the absolute error of a position of a distance-based localization system.*
- [matrix\\_t](#) [dist\\_based\\_get\\_distance\\_to\\_anchor](#) ([matrix\\_t](#) ref\_point[3], [matrix\\_t](#) point[3])  
*Computes the distance between a mobile station and a reference station of a distance-based localization system.*

### 10.10.1 Detailed Description

Functions of distance-based localization systems.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.10.2 Function Documentation

#### 10.10.2.1 dist\_based\_get\_absolute\_error()

```
void dist_based_get_absolute_error (
    matrix\_t value_arr[],
    matrix\_t approx_value_arr[],
    matrix\_t absolute_error_arr[],
    uint8\_t length )
```

Computes the absolute error of a position of a distance-based localization system.

Parameters

in	<a href="#">value_arr[]</a>	true position.
in	<a href="#">approx_value_arr[]</a>	approximate position of the mobile device.
in, out	<a href="#">absolute_error_arr[]</a>	includes the absolute error.
in	<a href="#">length</a>	arrays length.

Definition at line 29 of file [dist\\_based\\_position.c](#).

### 10.10.2.2 `dist_based_get_distance_to_anchor()`

```
matrix_t dist_based_get_distance_to_anchor (
    matrix_t ref_point[3],
    matrix_t point[3] )
```

Computes the distance between a mobile station and a reference station of a distance-based localization system.

#### Parameters

in	<i>ref_point</i>	three-dimensional coordinates of the reference stations.
in	<i>point</i>	three-dimensional coordinates of the mobile device.

#### Returns

the distance between the mobile station and the reference station.

Definition at line 45 of file `dist_based_position.c`.

References `matrix_t`, `vector_get_norm2()`, and `vector_sub()`.

## 10.11 `distance_based_test.c` File Reference

Examples of localization algorithms of distance-based positioning systems.

```
#include <stdio.h>
#include "trilateration.h"
#include "matrix.h"
#include "vector.h"
#include "dist_based_position.h"
```

### Functions

- void `distance_based_test` (void)  
*Example of a distance-based localization system.*

#### 10.11.1 Detailed Description

Examples of localization algorithms of distance-based positioning systems.

Localization algorithms examples using distance measures (see the [methods of distance-based](#) localization systems).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

## 10.11.2 Function Documentation

### 10.11.2.1 distance\_based\_test()

```
void distance_based_test (
    void )
```

Example of a distance-based localization system.

This example shows how to use the trilateration algorithm, whereby an estimated position is calculated based on the distance measurements.

Definition at line 32 of file distance\_based\_test.c.

References `matrix_t`, `trilateration2()`, and `vector_flex_print()`.

## 10.12 distance\_based\_test.h File Reference

Examples of localization algorithms of distance-based positioning systems.

### Functions

- void [distance\\_based\\_test](#) (void)  
*Example of a distance-based localization system.*

### 10.12.1 Detailed Description

Examples of localization algorithms of distance-based positioning systems.

Localization algorithms examples using distance measures (see the [methods of distance-based](#) localization systems).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

### 10.12.2 Function Documentation

### 10.12.2.1 distance\_based\_test()

```
void distance_based_test (
    void )
```

Example of a distance-based localization system.

This example shows how to use the trilateration algorithm, whereby an estimated position is calculated based on the distance measurements.

Definition at line 32 of file distance\_based\_test.c.

References `matrix_t`, `trilateration2()`, and `vector_flex_print()`.

## 10.13 DOP.c File Reference

Compute the Position Dilution of Precision (PDOP).

```
#include <math.h>
#include "DOP.h"
#include "moore_penrose_pseudo_inverse.h"
#include "matrix.h"
```

### Functions

- [matrix\\_t get\\_PDOP](#) (uint8\_t m, [matrix\\_t](#) ref\_Matrix[m][3], [matrix\\_t](#) true\_pos[m])  
*Compute the Position Dilution of Precision (PDOP).*

### 10.13.1 Detailed Description

Compute the Position Dilution of Precision (PDOP).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
Naouar Guerchali

### 10.13.2 Function Documentation

#### 10.13.2.1 get\_PDOP()

```
matrix_t get_PDOP (
    uint8_t m,
    matrix_t ref_Matrix[m][3],
    matrix_t true_pos[m] )
```

Compute the Position Dilution of Precision (PDOP).

## Parameters

in	<i>m</i>	number of the reference stations.
in	<i>ref_Matrix</i>	three-dimensional coordinates of the reference stations.
in	<i>true_pos</i>	three-dimensional coordinates of the mobile device.

## Returns

the PDOP-value.

Definition at line 30 of file DOP.c.

References `matrix_mul()`, `matrix_t`, `matrix_transpose()`, and `moore_penrose_get_pinv()`.

Referenced by `multipath_algo_own_norm_distr_test()`, and `pos_algos_common_test()`.

## 10.14 DOP.h File Reference

Compute the Position Dilution of Precision (PDOP).

```
#include <inttypes.h>
#include "matrix.h"
```

### Functions

- [matrix\\_t get\\_PDOP](#) (uint8\_t m, [matrix\\_t](#) ref\_Matrix[m][3], [matrix\\_t](#) true\_pos[m])  
*Compute the Position Dilution of Precision (PDOP).*

#### 10.14.1 Detailed Description

Compute the Position Dilution of Precision (PDOP).

##### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
Naouar Guerchali

#### 10.14.2 Function Documentation

##### 10.14.2.1 get\_PDOP()

```
matrix_t get_PDOP (
    uint8_t m,
    matrix_t ref_Matrix[m][3],
    matrix_t true_pos[m] )
```

Compute the Position Dilution of Precision (PDOP).

**Parameters**

in	<i>m</i>	number of the reference stations.
in	<i>ref_Matrix</i>	three-dimensional coordinates of the reference stations.
in	<i>true_pos</i>	three-dimensional coordinates of the mobile device.

**Returns**

the PDOP-value.

Definition at line 30 of file DOP.c.

References `matrix_mul()`, `matrix_t`, `matrix_transpose()`, and `moore_penrose_get_pinv()`.

Referenced by `multipath_algo_own_norm_distr_test()`, and `pos_algos_common_test()`.

## 10.15 fsolve.c File Reference

Solve multi-variant nonlinear equation systems.

```
#include "vector.h"
#include "matrix.h"
#include "fsolve.h"
#include "newton_raphson.h"
#include "damped_newton_raphson.h"
```

**Functions**

- `uint8_t fsolve` (`uint8_t f_length`, `uint8_t x0_length`, `vector_t x0_arr[]`, enum `NON_LIN_ALGORITHM` `algo`, `vector_t est_x_arr[]`, `void(*get_non_lin_sys)(vector_t x_arr[], vector_t f_vec[])`, `void(*get_jacobian)(vector_t x_arr[], matrix_t J[][x0_length])`)

*Solve systems of multi-variant nonlinear equations.*

### 10.15.1 Detailed Description

Solve multi-variant nonlinear equation systems.

The multi-variant nonlinear equation systems are solved using damped or the Newton–Raphson algorithms.

**Author**

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.15.2 Function Documentation

### 10.15.2.1 fsolve()

```
uint8_t fsolve (
    uint8_t f_length,
    uint8_t x0_length,
    vector_t x0_arr[],
    enum NON_LIN_ALGORITHM algo,
    vector_t est_x_arr[],
    void(*) (vector_t x_arr[], vector_t f_vec[]) get_non_lin_sys,
    void(*) (vector_t x_arr[], matrix_t J[][x0_length]) get_jacobian )
```

Solve systems of multi-variant nonlinear equations.

The user should provide pointers to non-linear equation systems and Jacobian functions. The user can choose between the damped or the Newton–Raphson algorithms.

#### Note

This function is generally implemented.

**Parameters**

in	<i>f_length</i>	length of the error functions vector.
in	<i>x0_length</i>	length of the start vector.
in	<i>x0_arr[]</i>	start vector.
in	<i>algo</i>	damped or the Newton–Raphson algorithm.
in, out	<i>est_x_arr[]</i>	estimated (solution) vector.
in	<i>(*get_non_lin_sys)</i>	pointer to non-linear equation systems.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

**Returns**

required iteration number.

Definition at line 29 of file fsolve.c.

References `damped_newton_raphson()`, `Damped_Newton_Raphson`, `Newton_Raphson`, and `newton_raphson()`.

Referenced by `fsolve_test()`.

## 10.16 fsolve.h File Reference

Solve multi-variant nonlinear equation systems.

**Enumerations**

- enum `NON_LIN_ALGORITHM` { `Newton_Raphson`, `Damped_Newton_Raphson` }  
Possible algorithms to solve multi-variant nonlinear equation systems.

**Functions**

- uint8\_t `fsolve` (uint8\_t f\_length, uint8\_t x0\_length, vector\_t x0\_arr[], enum `NON_LIN_ALGORITHM` algo, vector\_t est\_x\_arr[], void(\*get\_non\_lin\_sys)(vector\_t x\_arr[], vector\_t f\_vec[]), void(\*get\_jacobian)(vector\_t x\_arr[], matrix\_t J[][x0\_length]))  
Solve systems of multi-variant nonlinear equations.

### 10.16.1 Detailed Description

Solve multi-variant nonlinear equation systems.

The multi-variant nonlinear equation systems are solved using damped or the Newton–Raphson algorithms.

**Author**

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.16.2 Enumeration Type Documentation

#### 10.16.2.1 NON\_LIN\_ALGORITHM

enum `NON_LIN_ALGORITHM`

Possible algorithms to solve multi-variant nonlinear equation systems.

## Enumerator

Newton_Raphson	Newton–Raphson algorithm.
Damped_Newton_Raphson	Damped Newton–Raphson algorithm.

Definition at line 30 of file fsolve.h.

### 10.16.3 Function Documentation

#### 10.16.3.1 fsolve()

```
uint8_t fsolve (
    uint8_t f_length,
    uint8_t x0_length,
    vector_t x0_arr[],
    enum NON_LIN_ALGORITHM algo,
    vector_t est_x_arr[],
    void(*) (vector_t x_arr[], vector_t f_vec[]) get_non_lin_sys,
    void(*) (vector_t x_arr[], matrix_t J[][x0_length]) get_jacobian )
```

Solve systems of multi-variant nonlinear equations.

The user should provide pointers to non-linear equation systems and Jacobian functions. The user can choose between the damped or the Newton–Raphson algorithms.

#### Note

This function is generally implemented.

#### Parameters

in	<i>f_length</i>	length of the error functions vector.
in	<i>x0_length</i>	length of the start vector.
in	<i>x0_arr[]</i>	start vector.
in	<i>algo</i>	damped or the Newton–Raphson algorithm.
in, out	<i>est_x_arr[]</i>	estimated (solution) vector.
in	<i>(*get_non_lin_sys)</i>	pointer to non-linear equation systems.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

#### Returns

required iteration number.

Definition at line 29 of file fsolve.c.

References `damped_newton_raphson()`, `Damped_Newton_Raphson`, `Newton_Raphson`, and `newton_raphson()`.

Referenced by `fsolve_test()`.

## 10.17 fsolve\_test.c File Reference

Examples of solving non-linear equation systems.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <float.h>
#include "vector.h"
#include "matrix.h"
#include "damped_newton_raphson.h"
#include "newton_raphson.h"
#include "fsolve.h"
```

### Functions

- void [get\\_non\\_lin\\_sys\\_f1\\_](#) ([vector\\_t](#) x\_arr[], [vector\\_t](#) f1\_vec[])  
*The non-linear system  $f_1$ .*
- void [get\\_non\\_lin\\_sys\\_J1\\_](#) ([vector\\_t](#) x\_arr[], [matrix\\_t](#) J1[ ][2])  
*The Jacobian matrix of the non-linear system  $f_1$ .*
- void [get\\_non\\_lin\\_sys\\_f2\\_](#) ([vector\\_t](#) x\_arr[], [vector\\_t](#) f2\_vec[])  
*The non-linear system  $f_2$ .*
- void [get\\_non\\_lin\\_sys\\_J2\\_](#) ([vector\\_t](#) x\_arr[], [matrix\\_t](#) J2[ ][3])  
*The Jacobian matrix of the non-linear system  $f_2$ .*
- void [get\\_non\\_lin\\_sys\\_f3\\_](#) ([vector\\_t](#) x\_arr[], [vector\\_t](#) f3\_vec[])  
*The non-linear system  $f_3$ .*
- void [get\\_non\\_lin\\_sys\\_J3\\_](#) ([vector\\_t](#) x\_arr[], [matrix\\_t](#) J3[ ][3])  
*The Jacobian matrix of the non-linear system  $f_3$ .*
- void [fsolve\\_test](#) (void)  
*Examples of solving non-linear equation systems.*

### 10.17.1 Detailed Description

Examples of solving non-linear equation systems.

Solving non-linear equation systems examples (see [fsolve](#) functions).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.17.2 Function Documentation

#### 10.17.2.1 [get\\_non\\_lin\\_sys\\_f1\\_\(\)](#)

```
void get_non_lin_sys_f1_ (
    vector\_t x_arr[],
    vector\_t f1_vec[] )
```

The non-linear system  $f_1$ .

$$f_1(x_1, x_2) = \begin{bmatrix} x_1^3 + x_2 - 1 \\ x_2^3 - x_1 + 1 \end{bmatrix}$$

## Parameters

in	<code>x_arr[]</code>	pointer to the $x_1$ and $x_2$ values.
in	<code>f1_vec[]</code>	pointer to the $f_1$ function values.

Definition at line 54 of file fsolve\_test.c.

Referenced by fsolve\_test().

## 10.17.2.2 get\_non\_lin\_sys\_f2\_()

```
void get_non_lin_sys_f2_ (
    vector_t x_arr[],
    vector_t f2_vec[] )
```

The non-linear system  $f_2$ .

$$f_2(x_1, x_2, x_3) = \begin{bmatrix} 3x_1 - \cos(x_2 \times x_3) - \frac{1}{2} \\ x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.06 \\ \exp(-x_1 \times x_2) + 20x_3 + \frac{10\pi - 3}{3} \end{bmatrix}$$

## Parameters

in	<code>x_arr[]</code>	pointer to the $x_1$ , $x_2$ , and $x_3$ values.
in	<code>f2_vec[]</code>	pointer to the $f_2$ function values.

Definition at line 110 of file fsolve\_test.c.

References M\_PI.

Referenced by fsolve\_test().

## 10.17.2.3 get\_non\_lin\_sys\_f3\_()

```
void get_non_lin_sys_f3_ (
    vector_t x_arr[],
    vector_t f3_vec[] )
```

The non-linear system  $f_3$ .

$$f_3(x_1, x_2, x_3) = \begin{bmatrix} \exp(-x_1 x_2) + \log(x_1) - \exp(-2) \\ \exp(x_1) - \frac{\sqrt{x_3}}{x_1} - \exp(1) + 2 \\ x_1 + x_2 - x_2 x_3 + 5 \end{bmatrix}$$

**Parameters**

in	<code>x_arr[]</code>	pointer to the $x_1$ , $x_2$ , and $x_3$ values.
in	<code>f3_vec[]</code>	pointer to the $f_3$ function values.

Definition at line 174 of file `fsolve_test.c`.

Referenced by `fsolve_test()`.

**10.17.2.4 get\_non\_lin\_sys\_J1()**

```
void get_non_lin_sys_J1_ (
    vector_t x_arr[],
    matrix_t J1[][2] )
```

The Jacobian matrix of the non-linear system  $f_1$ .

$$J_1(x_1, x_2) = \begin{bmatrix} 3 \times x_1^2 & 1 \\ -1 & 3 \times x_2^2 \end{bmatrix}$$

## Parameters

in	<code>x_arr[]</code>	pointer to the $x_1$ and $x_2$ values.
in	<code>J1[][]</code>	pointer to the Jacobian function $J_1$ .

Definition at line 80 of file fsolve\_test.c.

Referenced by fsolve\_test().

## 10.17.2.5 get\_non\_lin\_sys\_J2\_()

```
void get_non_lin_sys_J2_ (
    vector_t x_arr[],
    matrix_t J2[][3] )
```

The Jacobian matrix of the non-linear system  $f_2$ .

$$J_2(x_1, x_2, x_3) = \begin{bmatrix} 3 & x_3 \sin(x_2 x_3) & x_2 \sin(x_2 x_3) \\ 2x_1 & -162x_2 - \frac{81}{5} & \cos(x_3) \\ -x_2 \exp(-x_1 x_2) & -x_1 \exp(-x_1 x_2) & 20 \end{bmatrix}$$

## Parameters

in	<code>x_arr[]</code>	pointer to the $x_1$ , $x_2$ , and $x_3$ values.
in	<code>J2[][]</code>	pointer to the Jacobian function $J_2$ .

Definition at line 139 of file fsolve\_test.c.

Referenced by fsolve\_test().

## 10.17.2.6 get\_non\_lin\_sys\_J3\_()

```
void get_non_lin_sys_J3_ (
    vector_t x_arr[],
    matrix_t J3[][3] )
```

The Jacobian matrix of the non-linear system  $f_3$ .

$$J_3(x_1, x_2, x_3) = \begin{bmatrix} \frac{1}{x_1} - x_2 \exp(-x_1 x_2) & -x_1 \exp(-x_1 x_2) & 0 \\ \exp(x_1) + \frac{\sqrt{x_3}}{x_1^2} & 0 & \frac{-1}{2x_1 \sqrt{x_3}} \\ 1 & 1 - x_3 & -x_2 \end{bmatrix}$$

**Parameters**

in	$x\_arr[]$	pointer to the $x_1$ , $x_2$ , and $x_3$ values.
in	$J3[][]$	pointer to the Jacobian function $J_3$ .

Definition at line 210 of file `fsolve_test.c`.

Referenced by `fsolve_test()`.

## 10.18 fsolve\_test.h File Reference

Examples of solving non-linear equation systems.

### Functions

- void [fsolve\\_test](#) (void)  
*Examples of solving non-linear equation systems.*

### 10.18.1 Detailed Description

Examples of solving non-linear equation systems.

Solving non-linear equation systems examples (see [fsolve](#) functions).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.19 givens\_test.c File Reference

Examples of the Givens algorithm.

```
#include <stdio.h>
#include "qr_givens.h"
#include "matrix.h"
```

### Functions

- void [givens\\_test](#) (void)  
*Examples of the Givens algorithm.*

### 10.19.1 Detailed Description

Examples of the Givens algorithm.

Givens algorithm examples (see [qr\\_givens.h](#) the "Givens" approach).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.20 givens\_test.h File Reference

Examples of the Givens algorithm.

### Functions

- void [givens\\_test](#) (void)  
*Examples of the Givens algorithm.*

### 10.20.1 Detailed Description

Examples of the Givens algorithm.

Givens algorithm examples (see [qr\\_givens.h](#) the "Givens" approach).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.21 householder\_test.c File Reference

Examples of the Householder algorithm.

```
#include <stdbool.h>
#include <stdio.h>
#include "matrix.h"
#include "qr_householder.h"
```

### Functions

- void [householder\\_test](#) (void)  
*Examples of the Householder algorithm.*

### 10.21.1 Detailed Description

Examples of the Householder algorithm.

Householder algorithm examples (see the [Householder](#) approach).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.22 `householder_test.h` File Reference

Examples of the Householder algorithm.

### Functions

- void [householder\\_test](#) (void)  
*Examples of the Householder algorithm.*

### 10.22.1 Detailed Description

Examples of the Householder algorithm.

Householder algorithm examples (see the [Householder](#) approach).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.23 `levenberg_marquardt.c` File Reference

Implement the Levenberg–Marquardt (LVM) algorithm.

```
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <inttypes.h>
#include "levenberg_marquardt.h"
#include "matrix.h"
#include "vector.h"
#include "solve.h"
#include "utils.h"
```

## Functions

- `matrix_t opt levenberg_marquardt_correction` (uint8\_t f\_length, uint8\_t n, `matrix_t` x\_vec[n], `matrix_t` data\_vec[f\_length], `matrix_t` mu, `matrix_t` s[n], void(\*get\_f\_error)(`vector_t` x\_vec[], `vector_t` data\_vec[], `vector_t` f\_vec[]), void(\*get\_jacobian)(`vector_t` x\_vec[], `matrix_t` J[][n]))  
*Implements the correction-function of the Levenberg–Marquardt (LVM) algorithm.*
- `uint8_t opt levenberg_marquardt` (uint8\_t f\_length, uint8\_t n, `vector_t` x0\_vec[n], `vector_t` data\_vec[f\_length], `matrix_t` eps, `matrix_t` tau, `matrix_t` beta0, `matrix_t` beta1, uint8\_t max\_iter\_num, `vector_t` est\_x\_vec[n], void(\*get\_f\_error)(`vector_t` x0\_vec[], `vector_t` data\_vec[], `vector_t` f\_vec[]), void(\*get\_jacobian)(`vector_t` x0\_vec[], `matrix_t` J[][n]))  
*Implements the Levenberg–Marquardt (LVM) algorithm.*
- `matrix_t opt levenberg_marquardt_get_mu0` (uint8\_t n, `matrix_t` tau, `matrix_t` JTJ[][n])  
*Compute the initial value  $\mu_0$  of the Levenberg–Marquardt (LVM) algorithm.*

### 10.23.1 Detailed Description

Implement the Levenberg–Marquardt (LVM) algorithm.

#### Note

This function is generally implemented.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
 Naouar Guerchali

### 10.23.2 Function Documentation

#### 10.23.2.1 opt levenberg\_marquardt()

```
uint8_t opt levenberg_marquardt (
    uint8_t f_length,
    uint8_t n,
    vector_t x0_vec[n],
    vector_t data_vec[f_length],
    matrix_t eps,
    matrix_t tau,
    matrix_t beta0,
    matrix_t beta1,
    uint8_t max_iter_num,
    vector_t est_x_vec[n],
    void(*) (vector_t x0_vec[], vector_t data_vec[], vector_t f_vec[]) get_f_error,
    void(*) (vector_t x0_vec[], matrix_t J[][n]) get_jacobian )
```

Implements the Levenberg–Marquardt (LVM) algorithm.

The user should provide pointers to the error and Jacobian functions.

#### Note

This function is generally implemented.

## Parameters

in	<i>f_length</i>	length of the error functions vector.
in	<i>n</i>	length of the start vector.
in	<i>x0_vec[]</i>	start vector.
in	<i>data_vec[]</i>	data vector.
in	<i>eps</i>	accuracy bound.
in	<i>tau</i>	$\tau$ factor.
in	<i>beta0</i>	$\beta_0$ factor.
in	<i>beta1</i>	$\beta_1$ factor.
in	<i>max_iter_num</i>	maximal iteration number of the LVM algorithm.
out	<i>est_x_vec[]</i>	estimated (optimized) vector.
in	<i>(*get_f_error)</i>	pointer to the error function.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

## Returns

required iteration number.

Definition at line 143 of file `levenberg_marquardt.c`.

References `matrix_add_to_diag()`, `matrix_mul_scalar()`, `matrix_t`, `matrix_trans_mul_itself()`, `matrix_trans_mul_vec()`, `opt_levenberg_marquardt_correction()`, `opt_levenberg_marquardt_get_mu0()`, `solve_householder()`, `vector_add()`, `vector_copy()`, `vector_get_norm2()`, and `vector_t`.

Referenced by `optimization_exponential_data_test()`, `optimization_sinusoidal_data_test()`, and `optimization_test()`.

10.23.2.2 `opt_levenberg_marquardt_correction()`

```
matrix_t opt_levenberg_marquardt_correction (
    uint8_t f_length,
    uint8_t n,
    matrix_t x_vec[n],
    matrix_t data_vec[f_length],
    matrix_t mu,
    matrix_t s[n],
    void(*) (vector_t x_vec[], vector_t data_vec[], vector_t f_vec[]) get_f_error,
    void(*) (vector_t x_vec[], matrix_t J[][n]) get_jacobian )
```

Implements the correction-function of the Levenberg–Marquardt (LVM) algorithm.

The user should provide pointers to the error and Jacobian functions.

## Note

This function is generally implemented.

## Parameters

in	<i>f_length</i>	length of the error functions vector.
in	<i>n</i>	length of the start vector.
in	<i>x_vec[]</i>	start vector.
in	<i>data_vec[]</i>	data vector.
in	<i>mu</i>	regularization parameter $\mu$ .
out	<i>s[]</i>	correction vector.
in	<i>(*get_f_error)</i>	pointer to the error function that calculates the matrix $J_f^T J_f$ .
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

## Returns

the parameter  $\rho_\mu$

Definition at line 53 of file levenberg\_marquardt.c.

References `matrix_add_to_diag()`, `matrix_mul_scalar()`, `matrix_mul_vec()`, `matrix_t`, `matrix_trans_mul_itself()`, `matrix_trans_mul_vec()`, `solve_householder()`, `vector_add()`, `vector_get_scalar_product()`, and `vector_t`.

Referenced by `opt levenberg_marquardt()`.

## 10.23.2.3 opt levenberg\_marquardt\_get\_mu0()

```
matrix_t opt levenberg_marquardt_get_mu0 (
    uint8_t n,
    matrix_t tau,
    matrix_t JTJ[] [n] )
```

Compute the initial value  $\mu_0$  of the Levenberg–Marquardt (LVM) algorithm.

The user should provide a pointer to the matrix  $J_f^T J_f$ .

## Parameters

in	<i>n</i>	column number of the matrix $J_f^T J_f$ .
in	<i>tau</i>	$\tau$ factor.
in	<i>JTJ[][]</i>	pointer to the matrix $J_f^T J_f$ .

## Returns

parameter  $\rho_\mu$

Definition at line 227 of file levenberg\_marquardt.c.

References `matrix_t`.

Referenced by `opt levenberg_marquardt()`.

## 10.24 levenberg\_marquardt.h File Reference

Implement the Levenberg–Marquardt (LVM) algorithm.

```
#include <inttypes.h>
#include "matrix.h"
#include "vector.h"
```

### Functions

- `uint8_t opt levenberg_marquardt` (`uint8_t f_length`, `uint8_t n`, `vector_t x0_vec[n]`, `vector_t data_vec[f_length]`, `matrix_t eps`, `matrix_t tau`, `matrix_t beta0`, `matrix_t beta1`, `uint8_t max_iter_num`, `vector_t est_x_vec[n]`, `void(*get_f_error)(vector_t x0_vec[], vector_t data_vec[], vector_t f_vec[], void(*get_jacobian)(vector_t x0_vec[], matrix_t J[][n]))`  
*Implements the Levenberg–Marquardt (LVM) algorithm.*
- `matrix_t opt levenberg_marquardt_get_mu0` (`uint8_t n`, `matrix_t tau`, `matrix_t J TJ[][n]`)  
*Compute the initial value  $\mu_0$  of the Levenberg–Marquardt (LVM) algorithm.*

### 10.24.1 Detailed Description

Implement the Levenberg–Marquardt (LVM) algorithm.

#### Note

This function is generally implemented.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
 Naouar Guerchali

### 10.24.2 Function Documentation

#### 10.24.2.1 opt levenberg\_marquardt()

```
uint8_t opt levenberg_marquardt (
    uint8_t f_length,
    uint8_t n,
    vector_t x0_vec[n],
    vector_t data_vec[f_length],
    matrix_t eps,
    matrix_t tau,
    matrix_t beta0,
    matrix_t beta1,
    uint8_t max_iter_num,
    vector_t est_x_vec[n],
    void(*) (vector_t x0_vec[], vector_t data_vec[], vector_t f_vec[]) get_f_error,
    void(*) (vector_t x0_vec[], matrix_t J[][n]) get_jacobian )
```

Implements the Levenberg–Marquardt (LVM) algorithm.

The user should provide pointers to the error and Jacobian functions.

#### Note

This function is generally implemented.

## Parameters

in	<i>f_length</i>	length of the error functions vector.
in	<i>n</i>	length of the start vector.
in	<i>x0_vec[]</i>	start vector.
in	<i>data_vec[]</i>	data vector.
in	<i>eps</i>	accuracy bound.
in	<i>tau</i>	$\tau$ factor.
in	<i>beta0</i>	$\beta_0$ factor.
in	<i>beta1</i>	$\beta_1$ factor.
in	<i>max_iter_num</i>	maximal iteration number of the LVM algorithm.
out	<i>est_x_vec[]</i>	estimated (optimized) vector.
in	<i>(*get_f_error)</i>	pointer to the error function.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

## Returns

required iteration number.

Definition at line 143 of file levenberg\_marquardt.c.

References `matrix_add_to_diag()`, `matrix_mul_scalar()`, `matrix_t`, `matrix_trans_mul_itself()`, `matrix_trans_mul_vec()`, `opt_levenberg_marquardt_correction()`, `opt_levenberg_marquardt_get_mu0()`, `solve_householder()`, `vector_add()`, `vector_copy()`, `vector_get_norm2()`, and `vector_t`.

Referenced by `optimization_exponential_data_test()`, `optimization_sinusoidal_data_test()`, and `optimization_test()`.

## 10.24.2.2 opt\_levenberg\_marquardt\_get\_mu0()

```
matrix_t opt_levenberg_marquardt_get_mu0 (
    uint8_t n,
    matrix_t tau,
    matrix_t JTJ[ ][n] )
```

Compute the initial value  $\mu_0$  of the Levenberg–Marquardt (LVM) algorithm.

The user should provide a pointer to the matrix  $J_f^T J_f$ .

## Parameters

in	<i>n</i>	column number of the matrix $J_f^T J_f$ .
in	<i>tau</i>	$\tau$ factor.
in	<i>JTJ[ ][ ]</i>	pointer to the matrix $J_f^T J_f$ .

## Returns

parameter  $\rho_\mu$

Definition at line 227 of file levenberg\_marquardt.c.

References `matrix_t`.

Referenced by `opt_levenberg_marquardt()`.

## 10.25 loc\_gauss\_newton.c File Reference

Implement the Gauss–Newton algorithm.

```
#include <stdio.h>
#include "utils.h"
#include "matrix.h"
#include "vector.h"
#include "moore_penrose_pseudo_inverse.h"
```

### Functions

- `uint8_t loc_gauss_newton` (`uint8_t` ref\_points\_num, `matrix_t` ref\_points\_matrix[ref\_points\_num][3], `vector_t` start\_pos[3], `matrix_t` measured\_data\_vec[ref\_points\_num], `matrix_t` eps, `matrix_t` fmin, `uint8_t` max\_iter\_num, `vector_t` est\_pos[3], `void`(\*f\_i)(`uint8_t` ref\_point\_num, `matrix_t` ref\_point\_mat[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` d\_vec[], `matrix_t` f\_vec[]), `void`(\*jacobian\_get\_JTJ)(`uint8_t` ref\_points\_num, `matrix_t` ref\_point\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTJ[3][3]), `void`(\*jacobian\_get\_JTf)(`uint8_t` ref\_points\_num, `matrix_t` ref\_point\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTf[3]))

*Implements the modified Gauss–Newton algorithm.*

### 10.25.1 Detailed Description

Implement the Gauss–Newton algorithm.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

### 10.25.2 Function Documentation

### 10.25.2.1 loc\_gauss\_newton()

```
uint8_t loc_gauss_newton (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ref_points_num][3],
    vector_t start_pos[3],
    matrix_t measured_data_vec[ref_points_num],
    matrix_t eps,
    matrix_t fmin,
    uint8_t max_iter_num,
    vector_t est_pos[3],
    void(*) (uint8_t ref_point_num, matrix_t ref_point_mat[ref_points_num][3], matrix_t
point[3], matrix_t d_vec[], matrix_t f_vec[]) f_i,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTJ[3][3]) jacobian_get_JTJ,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTf[3]) jacobian_get_JTf )
```

Implements the modified Gauss–Newton algorithm.

The user should provide pointers to the error and Jacobian functions.

#### Note

This function is optimized for localization algorithms.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>start_pos</i> []	start (approximate) position.
in	<i>measured_data_vec</i> []	pointer to the measured data.
in	<i>eps</i>	accuracy bound.
in	<i>fmin</i>	termination tolerance on the error function.
in	<i>max_iter_num</i>	maximal iteration number of the Gauss-Newton algorithm.
in, out	<i>est_pos</i> []	estimated (optimized) position.
in	( <i>*f_i</i> )	pointer to the error function.
in	( <i>*jacobian_get_JTJ</i> )	pointer to the function that calculates the $J_f^T J_f$ matrix.
in	( <i>*jacobian_get_JTf</i> )	pointer to the function that calculates the $J_f^T \vec{f}$ vector.

#### Returns

required iteration number.

Definition at line 30 of file loc\_gauss\_newton.c.

References `matrix_mul_vec()`, `matrix_t`, `moore_penrose_get_pinv()`, `utils_max()`, `vector_copy()`, `vector_get_↵`  
`euclidean_distance()`, `vector_get_norm2()`, and `vector_sub()`.

Referenced by `position_optimization_test()`.

## 10.26 loc\_gauss\_newton.h File Reference

Implement the Gauss–Newton algorithm for position optimization.

```
#include <inttypes.h>
#include "matrix.h"
#include "vector.h"
```

### Functions

- `uint8_t loc_gauss_newton` (`uint8_t ref_points_num`, `matrix_t ref_points_matrix[ref_points_num][3]`, `vector_t start_pos[3]`, `matrix_t measured_data_vec[ref_points_num]`, `matrix_t eps`, `matrix_t fmin`, `uint8_t max_iter_num`, `vector_t est_pos[3]`, `void(*f_i)(uint8_t ref_point_num, matrix_t ref_point_mat[ref_points_num][3], matrix_t point[3], matrix_t d_vec[], matrix_t f_vec[])`, `void(*jacobian_get_JTJ)(uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3], matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTJ[3][3])`, `void(*jacobian_get_JTf)(uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3], matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTf[3])`)

*Implements the modified Gauss–Newton algorithm.*

### 10.26.1 Detailed Description

Implement the Gauss–Newton algorithm for position optimization.

#### Note

This function is adapted for localization algorithms.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
 Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

### 10.26.2 Function Documentation

#### 10.26.2.1 loc\_gauss\_newton()

```
uint8_t loc_gauss_newton (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ref_points_num][3],
    vector_t start_pos[3],
    matrix_t measured_data_vec[ref_points_num],
    matrix_t eps,
    matrix_t fmin,
    uint8_t max_iter_num,
    vector_t est_pos[3],
    void(*) (uint8_t ref_point_num, matrix_t ref_point_mat[ref_points_num][3], matrix_t
point[3], matrix_t d_vec[], matrix_t f_vec[]) f_i,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTJ[3][3]) jacobian_get_JTJ,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTf[3]) jacobian_get_JTf )
```

Implements the modified Gauss–Newton algorithm.

The user should provide pointers to the error and Jacobian functions.

**Note**

This function is optimized for localization algorithms.

**Parameters**

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>start_pos</i> []	start (approximate) position.
in	<i>measured_data_vec</i> []	pointer to the measured data.
in	<i>eps</i>	accuracy bound.
in	<i>fmin</i>	termination tolerance on the error function.
in	<i>max_iter_num</i>	maximal iteration number of the Gauss-Newton algorithm.
in, out	<i>est_pos</i> []	estimated (optimized) position.
in	( <i>*f_i</i> )	pointer to the error function.
in	( <i>*jacobian_get_JTJ</i> )	pointer to the function that calculates the $J_f^T J_f$ matrix.
in	( <i>*jacobian_get_JTf</i> )	pointer to the function that calculates the $J_f^T \vec{f}$ vector.

**Returns**

required iteration number.

Definition at line 30 of file loc\_gauss\_newton.c.

References `matrix_mul_vec()`, `matrix_t`, `moore_penrose_get_pinv()`, `utils_max()`, `vector_copy()`, `vector_get_euclidean_distance()`, `vector_get_norm2()`, and `vector_sub()`.

Referenced by `position_optimization_test()`.

## 10.27 loc\_levenberg\_marquardt.c File Reference

Implement the Levenberg–Marquardt (LVM) algorithm.

```
#include <stdio.h>
#include <math.h>
#include <float.h>
#include <inttypes.h>
#include "matrix.h"
#include "vector.h"
#include "solve.h"
#include "utils.h"
#include "loc_levenberg_marquardt.h"
```

## Functions

- `matrix_t loc_levenberg_marquardt_correction` (uint8\_t ref\_points\_num, `matrix_t` ref\_points\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` measured\_data\_vec[ref\_points\_num], `matrix_t` mu, `matrix_t` s[3], void(\*f\_i)(uint8\_t ref\_point\_num, `matrix_t` ref\_point\_mat[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` d\_vec[], `matrix_t` f\_vec[]), void(\*jacobian\_get\_JTJ)(uint8\_t ref\_points\_num, `matrix_t` ref\_point\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTJ[3][3]), void(\*jacobian\_get\_JTf)(uint8\_t ref\_points\_num, `matrix_t` ref\_point\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTf[3]), void(\*jacobian\_get\_J\_mul\_s)(uint8\_t ref\_points\_num, `matrix_t` ref\_point\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` s[3], `matrix_t` J\_s[ref\_points\_num]))

*Implements the correction-function of the Levenberg–Marquardt (LVM) algorithm.*

- `uint8_t loc_levenberg_marquardt` (uint8\_t ref\_points\_num, `matrix_t` ref\_points\_matrix[ref\_points\_num][3], `matrix_t` start\_pos[3], `matrix_t` measured\_data\_vec[ref\_points\_num], `matrix_t` eps, `matrix_t` tau, `matrix_t` beta0, `matrix_t` beta1, uint8\_t max\_iter\_num, `matrix_t` est\_pos[3], void(\*f\_i)(uint8\_t ref\_points\_num, `matrix_t` ref\_point\_mat[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` d\_vec[], `matrix_t` f\_vec[]), void(\*jacobian\_get\_JTJ)(uint8\_t ref\_points\_num, `matrix_t` ref\_point\_matrix[ ][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTJ[3][3]), void(\*jacobian\_get\_JTf)(uint8\_t ref\_points\_num, `matrix_t` ref\_point\_matrix[ ][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTf[3]), void(\*jacobian\_get\_J\_mul\_s)(uint8\_t ref\_points\_num, `matrix_t` ref\_point\_matrix[ ][3], `matrix_t` point[3], `matrix_t` s[3], `matrix_t` J\_s[ref\_points\_num]))

*Implements the Levenberg–Marquardt (LVM) algorithm.*

- `matrix_t loc_levenberg_marquardt_get_mu0` (`matrix_t` tau, `matrix_t` JTJ[3][3])

*Compute the initial value  $\mu_0$  of the Levenberg–Marquardt (LVM) algorithm.*

- `void loc_levenberg_marquardt_get_JTJ_mu2_l` (uint8\_t ref\_points\_num, `matrix_t` ref\_points\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` measured\_data\_vec[ref\_points\_num], `matrix_t` mu, `matrix_t` JTJ\_mu2\_l[3][3], void(\*jacobian\_get\_JTJ)(uint8\_t ref\_points\_num, `matrix_t` ref\_point\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTJ[3][3]))

*Compute the matrix  $J_f^T J_f + \mu^{(i)^2} I$ .*

### 10.27.1 Detailed Description

Implement the Levenberg–Marquardt (LVM) algorithm.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

Naouar Guerchali

### 10.27.2 Function Documentation

#### 10.27.2.1 loc\_levenberg\_marquardt()

```
uint8_t loc_levenberg_marquardt (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ref_points_num][3],
    matrix_t start_pos[3],
    matrix_t measured_data_vec[ref_points_num],
    matrix_t eps,
    matrix_t tau,
    matrix_t beta0,
```

```

    matrix_t beta1,
    uint8_t max_iter_num,
    matrix_t est_pos[3],
    void(*) (uint8_t ref_points_num, matrix_t ref_point_mat[ref_points_num][3], matrix_t
point[3], matrix_t d_vec[], matrix_t f_vec[]) f_i,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[][3], matrix_t point[3],
matrix_t data_vec[ref_points_num], matrix_t JTJ[3][3]) jacobian_get_JTJ,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[][3], matrix_t point[3],
matrix_t data_vec[ref_points_num], matrix_t JTf[3]) jacobian_get_JTf,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[][3], matrix_t point[3],
matrix_t s[3], matrix_t J_s[ref_points_num]) jacobian_get_J_mul_s )

```

Implements the Levenberg–Marquardt (LVM) algorithm.

The user should provide pointers to the error and Jacobian functions.

#### Note

This function is optimized for localization algorithms.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>start_pos</i> []	start (approximate) position.
in	<i>measured_data_vec</i> []	pointer to the measured data.
in	<i>eps</i>	accuracy bound.
in	<i>tau</i>	$\tau$ factor.
in	<i>beta0</i>	$\beta_0$ factor.
in	<i>beta1</i>	$\beta_1$ factor.
in	<i>max_iter_num</i>	maximal iteration number of the LVM algorithm.
out	<i>est_pos</i> []	estimated (optimized) position.
in	( <i>*f_i</i> )	pointer to the error function.
in	( <i>*jacobian_get_JTJ</i> )	pointer to the function that calculates the matrix $J_f^T J_f$ .
in	( <i>*jacobian_get_JTf</i> )	pointer to the function that calculates the vector $J_f^T \vec{f}$ .
in	( <i>*jacobian_get_J_mul_s</i> )	pointer to the function that calculates the vector $J_f^T \vec{s}$ .

#### Returns

required iteration number.

Definition at line 126 of file loc\_levenberg\_marquardt.c.

References loc\_levenberg\_marquardt\_correction(), loc\_levenberg\_marquardt\_get\_JTJ\_mu2\_I(), loc\_levenberg\_marquardt\_get\_mu0(), matrix\_mul\_scalar(), matrix\_t, solve\_householder(), vector\_add(), vector\_copy(), and vector\_get\_norm2().

Referenced by multipath\_algo\_own\_norm\_distr\_test(), and position\_optimization\_test().

### 10.27.2.2 loc\_levenberg\_marquardt\_correction()

```
matrix_t loc_levenberg_marquardt_correction (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t measured_data_vec[ref_points_num],
    matrix_t mu,
    matrix_t s[3],
    void(*) (uint8_t ref_point_num, matrix_t ref_point_mat[ref_points_num][3], matrix_t
point[3], matrix_t d_vec[], matrix_t f_vec[]) f_i,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTJ[3][3]) jacobian_get_JTJ,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTf[3]) jacobian_get_JTf,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
matrix_t point[3], matrix_t s[3], matrix_t J_s[ref_points_num]) jacobian_get_J_mul_s )
```

Implements the correction-function of the Levenberg–Marquardt (LVM) algorithm.

The user should provide pointers to the error and Jacobian functions.

#### Note

This function is optimized for localization algorithms.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix[ ][ ]</i>	three-dimensional coordinates of the reference stations.
in	<i>point[ ]</i>	a position to adjust.
in	<i>measured_data_vec[ ]</i>	pointer to the measured data.
in	<i>mu</i>	regularization parameter $\mu$ .
out	<i>s[ ]</i>	correction vector.
in	<i>(*f_i)</i>	pointer to the error function.
in	<i>(*jacobian_get_JTJ)</i>	pointer to the function that calculates the matrix $J_f^T J_f$ .
in	<i>(*jacobian_get_JTf)</i>	pointer to the function that calculates the vector $J_f^T \vec{f}$ .
in	<i>(*jacobian_get_J_mul_s)</i>	pointer to the function that calculates the vector $J_f^T \vec{s}$ .

#### Returns

the parameter  $\rho_\mu$

Definition at line 34 of file loc\_levenberg\_marquardt.c.

References loc\_levenberg\_marquardt\_get\_JTJ\_mu2\_l(), matrix\_mul\_scalar(), matrix\_t, solve\_householder(), vector\_add(), and vector\_get\_scalar\_product().

Referenced by loc\_levenberg\_marquardt().

### 10.27.2.3 loc\_levenberg\_marquardt\_get\_JTJ\_mu2\_I()

```
void loc_levenberg_marquardt_get_JTJ_mu2_I (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t measured_data_vec[ref_points_num],
    matrix_t mu,
    matrix_t JTJ_mu2_I[3][3],
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTJ[3][3]) jacobian_get_JTJ )
```

Compute the matrix  $J_f^T J_f + \mu^{(i)^2} I$ .

#### Note

This function is optimized for localization algorithms.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	a position to adjust.
in	<i>measured_data_vec</i> []	pointer to the measured data.
in	<i>mu</i>	$\rho_\mu$ -parameter
out	<i>JTJ_mu2_I</i> [][]	includes the matrix $J_f^T J_f + \mu^{(i)^2} I$ .
in	<i>(*jacobian_get_JTJ)</i>	pointer to the function that calculates the matrix $J_f^T J_f$ .

Definition at line 236 of file loc\_levenberg\_marquardt.c.

Referenced by loc\_levenberg\_marquardt(), and loc\_levenberg\_marquardt\_correction().

### 10.27.2.4 loc\_levenberg\_marquardt\_get\_mu0()

```
matrix_t loc_levenberg_marquardt_get_mu0 (
    matrix_t tau,
    matrix_t JTJ[3][3] )
```

Compute the initial value  $\mu_0$  of the Levenberg–Marquardt (LVM) algorithm.

The user should provide a pointer to the matrix  $J_f^T J_f$ .

#### Parameters

in	<i>tau</i>	$\tau$ factor.
in	<i>JTJ</i> [][]	pointer to the matrix $J_f^T J_f$ .

## Returns

$\rho_\mu$ -parameter

Definition at line 223 of file loc\_levenberg\_marquardt.c.

References matrix\_t.

Referenced by loc\_levenberg\_marquardt().

## 10.28 loc\_levenberg\_marquardt.h File Reference

Implement the Levenberg–Marquardt (LVM) algorithm for position optimization.

```
#include <inttypes.h>
#include "matrix.h"
```

### Functions

- `uint8_t loc_levenberg_marquardt` (`uint8_t` ref\_points\_num, `matrix_t` ref\_points\_matrix[ref\_points\_num][3], `matrix_t` start\_pos[3], `matrix_t` measured\_data\_vec[ref\_points\_num], `matrix_t` eps, `matrix_t` tau, `matrix_t` beta0, `matrix_t` beta1, `uint8_t` max\_iter\_num, `matrix_t` est\_pos[3], `void`(\*f\_i)(`uint8_t` ref\_points\_num, `matrix_t` ref\_point\_mat[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` d\_vec[], `matrix_t` f\_vec[]), `void`(\*jacobian\_get\_JTJ)(`uint8_t` ref\_points\_num, `matrix_t` ref\_point\_matrix[][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTJ[3][3]), `void`(\*jacobian\_get\_JTf)(`uint8_t` ref\_points\_num, `matrix_t` ref\_point\_matrix[][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTf[3]), `void`(\*jacobian\_get\_J\_mul\_s)(`uint8_t` ref\_points\_num, `matrix_t` ref\_point\_matrix[][3], `matrix_t` point[3], `matrix_t` s[3], `matrix_t` J\_s[ref\_points\_num]))

*Implements the Levenberg–Marquardt (LVM) algorithm.*

- `matrix_t loc_levenberg_marquardt_correction` (`uint8_t` ref\_points\_num, `matrix_t` ref\_points\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` measured\_data\_vec[ref\_points\_num], `matrix_t` mu, `matrix_t` s[3], `void`(\*f\_i)(`uint8_t` ref\_point\_num, `matrix_t` ref\_point\_mat[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` d\_vec[], `matrix_t` f\_vec[]), `void`(\*jacobian\_get\_JTJ)(`uint8_t` ref\_points\_num, `matrix_t` ref\_point\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTJ[3][3]), `void`(\*jacobian\_get\_JTf)(`uint8_t` ref\_points\_num, `matrix_t` ref\_point\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTf[3]), `void`(\*jacobian\_get\_J\_mul\_s)(`uint8_t` ref\_points\_num, `matrix_t` ref\_point\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` s[3], `matrix_t` J\_s[ref\_points\_num]))

*Implements the correction-function of the Levenberg–Marquardt (LVM) algorithm.*

- `matrix_t loc_levenberg_marquardt_get_mu0` (`matrix_t` tau, `matrix_t` JTJ[3][3])

*Compute the initial value  $\mu_0$  of the Levenberg–Marquardt (LVM) algorithm.*

- `void loc_levenberg_marquardt_get_JTJ_mu2_l` (`uint8_t` ref\_points\_num, `matrix_t` ref\_points\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` measured\_data\_vec[ref\_points\_num], `matrix_t` mu, `matrix_t` JTJ\_mu2\_l[3][3], `void`(\*jacobian\_get\_JTJ)(`uint8_t` ref\_points\_num, `matrix_t` ref\_point\_matrix[ref\_points\_num][3], `matrix_t` point[3], `matrix_t` data\_vec[ref\_points\_num], `matrix_t` JTJ[3][3]))

*Compute the matrix  $J_f^T J_f + \mu^{(i)} I$ .*

### 10.28.1 Detailed Description

Implement the Levenberg–Marquardt (LVM) algorithm for position optimization.

#### Note

This function is adapted for localization algorithms.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
Naouar Guerchali

## 10.28.2 Function Documentation

### 10.28.2.1 loc\_levenberg\_marquardt()

```
uint8_t loc_levenberg_marquardt (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ref_points_num][3],
    matrix_t start_pos[3],
    matrix_t measured_data_vec[ref_points_num],
    matrix_t eps,
    matrix_t tau,
    matrix_t beta0,
    matrix_t beta1,
    uint8_t max_iter_num,
    matrix_t est_pos[3],
    void(*) (uint8_t ref_points_num, matrix_t ref_point_mat[ref_points_num][3], matrix_t
point[3], matrix_t d_vec[], matrix_t f_vec[]) f_i,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[][3], matrix_t point[3],
matrix_t data_vec[ref_points_num], matrix_t JTJ[3][3]) jacobian_get_JTJ,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[][3], matrix_t point[3],
matrix_t data_vec[ref_points_num], matrix_t JTf[3]) jacobian_get_JTf,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[][3], matrix_t point[3],
matrix_t s[3], matrix_t J_s[ref_points_num]) jacobian_get_J_mul_s )
```

Implements the Levenberg–Marquardt (LVM) algorithm.

The user should provide pointers to the error and Jacobian functions.

#### Note

This function is optimized for localization algorithms.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix</i> [[ ]]	three-dimensional coordinates of the reference stations.
in	<i>start_pos</i> [ ]	start (approximate) position.
in	<i>measured_data_vec</i> [ ]	pointer to the measured data.
in	<i>eps</i>	accuracy bound.
in	<i>tau</i>	$\tau$ factor.
in	<i>beta0</i>	$\beta_0$ factor.
in	<i>beta1</i>	$\beta_1$ factor.
in	<i>max_iter_num</i>	maximal iteration number of the LVM algorithm.
out	<i>est_pos</i> [ ]	estimated (optimized) position.
in	( <i>*f_i</i> )	pointer to the error function.
in	( <i>*jacobian_get_JTJ</i> )	pointer to the function that calculates the matrix $J_f^T J_f$ .
in	( <i>*jacobian_get_JTf</i> )	pointer to the function that calculates the vector $J_f^T \vec{f}$ .
in	( <i>*jacobian_get_J_mul_s</i> )	pointer to the function that calculates the vector $J_f^T \vec{s}$ .

**Returns**

required iteration number.

Definition at line 126 of file `loc_levenberg_marquardt.c`.

References `loc_levenberg_marquardt_correction()`, `loc_levenberg_marquardt_get_JTJ_mu2_l()`, `loc_levenberg_marquardt_get_mu0()`, `matrix_mul_scalar()`, `matrix_t`, `solve_householder()`, `vector_add()`, `vector_copy()`, and `vector_get_norm2()`.

Referenced by `multipath_algo_own_norm_distr_test()`, and `position_optimization_test()`.

**10.28.2.2 loc\_levenberg\_marquardt\_correction()**

```
matrix_t loc_levenberg_marquardt_correction (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t measured_data_vec[ref_points_num],
    matrix_t mu,
    matrix_t s[3],
    void(*) (uint8_t ref_point_num, matrix_t ref_point_mat[ref_points_num][3], matrix_t
point[3], matrix_t d_vec[], matrix_t f_vec[]) f_i,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTJ[3][3]) jacobian_get_JTJ,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTf[3]) jacobian_get_JTf,
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
matrix_t point[3], matrix_t s[3], matrix_t J_s[ref_points_num]) jacobian_get_J_mul_s )
```

Implements the correction-function of the Levenberg–Marquardt (LVM) algorithm.

The user should provide pointers to the error and Jacobian functions.

**Note**

This function is optimized for localization algorithms.

**Parameters**

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	a position to adjust.
in	<i>measured_data_vec</i> []	pointer to the measured data.
in	<i>mu</i>	regularization parameter $\mu$ .
out	<i>s</i> []	correction vector.
in	<i>(*f_i)</i>	pointer to the error function.
in	<i>(*jacobian_get_JTJ)</i>	pointer to the function that calculates the matrix $J_f^T J_f$ .
in	<i>(*jacobian_get_JTf)</i>	pointer to the function that calculates the vector $J_f^T \vec{f}$ .
in	<i>(*jacobian_get_J_mul_s)</i>	pointer to the function that calculates the vector $J_f^T \vec{s}$ .

**Returns**

the parameter  $\rho_\mu$

Definition at line 34 of file loc\_levenberg\_marquardt.c.

References loc\_levenberg\_marquardt\_get\_JTJ\_mu2\_I(), matrix\_mul\_scalar(), matrix\_t, solve\_householder(), vector\_add(), and vector\_get\_scalar\_product().

Referenced by loc\_levenberg\_marquardt().

**10.28.2.3 loc\_levenberg\_marquardt\_get\_JTJ\_mu2\_I()**

```
void loc_levenberg_marquardt_get_JTJ_mu2_I (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t measured_data_vec[ref_points_num],
    matrix_t mu,
    matrix_t JTJ_mu2_I[3][3],
    void(*) (uint8_t ref_points_num, matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3], matrix_t data_vec[ref_points_num], matrix_t JTJ[3][3]) jacobian_get_JTJ )
```

Compute the matrix  $J_f^T J_f + \mu^{(i)^2} I$ .

**Note**

This function is optimized for localization algorithms.

## Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix</i> [[ ]]	three-dimensional coordinates of the reference stations.
in	<i>point</i> [ ]	a position to adjust.
in	<i>measured_data_vec</i> [ ]	pointer to the measured data.
in	<i>mu</i>	$\rho_\mu$ -parameter
out	<i>JTJ_mu2_l</i> [[ ]]	includes the matrix $J_f^T J_f + \mu^{(i)^2} I$ .
in	<i>(*jacobian_get_JTJ)</i>	pointer to the function that calculates the matrix $J_f^T J_f$ .

Definition at line 236 of file `loc_levenberg_marquardt.c`.

Referenced by `loc_levenberg_marquardt()`, and `loc_levenberg_marquardt_correction()`.

#### 10.28.2.4 loc\_levenberg\_marquardt\_get\_mu0()

```
matrix_t loc_levenberg_marquardt_get_mu0 (
    matrix_t tau,
    matrix_t JTJ[3][3] )
```

Compute the initial value  $\mu_0$  of the Levenberg–Marquardt (LVM) algorithm.

The user should provide a pointer to the matrix  $J_f^T J_f$ .

## Parameters

in	<i>tau</i>	$\tau$ factor.
in	<i>JTJ</i> [][]	pointer to the matrix $J_f^T J_f$ .

## Returns

$\rho_\mu$ -parameter

Definition at line 223 of file loc\_levenberg\_marquardt.c.

References matrix\_t.

Referenced by loc\_levenberg\_marquardt().

## 10.29 lu\_decomp.c File Reference

Computes the LU decomposition of the matrix.

```
#include <float.h>
#include <stdio.h>
#include "matrix.h"
#include "vector.h"
```

### Functions

- uint8\_t [lu\\_decomp](#) (uint8\_t n, [matrix\\_t](#) A[][n], [matrix\\_t](#) L[][n], [matrix\\_t](#) P[][n])  
*Computes the LU decomposition of the matrix.*

### 10.29.1 Detailed Description

Computes the LU decomposition of the matrix.

Computes the permutation matrix P such that:  $A = P^*L*U$ , where L is a lower triangular matrix and U is an upper triangular matrix. It implements the Gaussian Elimination (GE) with pivoting algorithm.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.29.2 Function Documentation

### 10.29.2.1 lu\_decomp()

```
uint8_t lu_decomp (
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t L[ ][n],
    matrix_t P[ ][n] )
```

Computes the LU decomposition of the matrix.

Computes the permutation matrix P such that:  $A = P' * L * U$ , where L is a lower triangular matrix and U is an upper triangular matrix. It implements the Gaussian Elimination with pivoting algorithm.

#### Note

Matrix U is stored in the matrix A.

#### Parameters

in	<i>n</i>	column number of the matrix.
in, out	<i>A[ ][ ]</i>	pointer to the matrices A and U.
out	<i>L[ ][ ]</i>	pointer to the L matrix.
out	<i>P[ ][ ]</i>	pointer to the P matrix.

#### Returns

the number of changes by computing the LU decomposition.

Definition at line 31 of file lu\_decomp.c.

References `matrix_get_abs_max_elem_and_index_in_part_column()`, `matrix_get_diag_mat()`, `matrix_part_swap_rows()`, `matrix_swap_rows()`, and `matrix_t`.

Referenced by `lu_decomp_test()`, and `solve_lu_decomp()`.

## 10.30 lu\_decomp.h File Reference

Computes the LU decomposition of the matrix.

```
#include "matrix.h"
```

### Functions

- `uint8_t lu_decomp (uint8_t n, matrix_t A[ ][n], matrix_t L[ ][n], matrix_t P[ ][n])`  
Computes the LU decomposition of the matrix.

### 10.30.1 Detailed Description

Computes the LU decomposition of the matrix.

Computes the permutation matrix  $P$  such that:  $A = P'L*U$ , where  $L$  is a lower triangular matrix and  $U$  is an upper triangular matrix. It implements the Gaussian Elimination (GE) with pivoting algorithm.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.30.2 Function Documentation

#### 10.30.2.1 lu\_decomp()

```
uint8_t lu_decomp (
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t L[ ][n],
    matrix_t P[ ][n] )
```

Computes the LU decomposition of the matrix.

Computes the permutation matrix  $P$  such that:  $A = P'L*U$ , where  $L$  is a lower triangular matrix and  $U$  is an upper triangular matrix. It implements the Gaussian Elimination with pivoting algorithm.

#### Note

Matrix  $U$  is stored in the matrix  $A$ .

#### Parameters

in	$n$	column number of the matrix.
in, out	$A[ ][ ]$	pointer to the matrices $A$ and $U$ .
out	$L[ ][ ]$	pointer to the $L$ matrix.
out	$P[ ][ ]$	pointer to the $P$ matrix.

#### Returns

the number of changes by computing the LU decomposition.

Definition at line 31 of file lu\_decomp.c.

References [matrix\\_get\\_abs\\_max\\_elem\\_and\\_index\\_in\\_part\\_column\(\)](#), [matrix\\_get\\_diag\\_mat\(\)](#), [matrix\\_part\\_swap\\_rows\(\)](#), [matrix\\_swap\\_rows\(\)](#), and [matrix\\_t](#).

Referenced by [lu\\_decomp\\_test\(\)](#), and [solve\\_lu\\_decomp\(\)](#).

## 10.31 lu\_decomp\_test.c File Reference

Examples of the LU algorithm with pivoting.

```
#include <stdio.h>
#include "matrix.h"
#include "lu_decomp.h"
```

### Functions

- void [lu\\_decomp\\_test](#) (void)  
*Examples of the LU algorithm with pivoting.*

#### 10.31.1 Detailed Description

Examples of the LU algorithm with pivoting.

LU algorithm with pivoting examples (see the [Gaussian Elimination with pivoting](#) approach).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.32 lu\_decomp\_test.h File Reference

Examples of the LU algorithm with pivoting.

### Functions

- void [lu\\_decomp\\_test](#) (void)  
*Examples of the LU algorithm with pivoting.*

#### 10.32.1 Detailed Description

Examples of the LU algorithm with pivoting.

LU algorithm with pivoting examples (see the [Gaussian Elimination with pivoting](#) approach).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.33 magnetic\_based\_fi.c File Reference

Error function of DC-pulsed, magnetic localization system.

```
#include <math.h>
#include <inttypes.h>
#include <magnetic_based_position.h>
#include "matrix.h"
```

### Functions

- void [magnetic\\_based\\_f\\_i](#) (uint8\_t ref\_points\_num, [matrix\\_t](#) ref\_points\_matrix[ ][3], [matrix\\_t](#) point[], [matrix\\_t](#) Bi\_vec[], [matrix\\_t](#) f\_vec[])

*Defines the error function of a magnetic-based localization system.*

### 10.33.1 Detailed Description

Error function of DC-pulsed, magnetic localization system.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

Naouar Guerchali

### 10.33.2 Function Documentation

#### 10.33.2.1 magnetic\_based\_f\_i()

```
void magnetic_based_f_i (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ ][3],
    matrix_t point[],
    matrix_t Bi_vec[],
    matrix_t f_vec[] )
```

Defines the error function of a magnetic-based localization system.

This error function is related to multiple reference stations.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix[ ][ ]</i>	three-dimensional coordinates of the reference stations.
in	<i>point[ ]</i>	three-dimensional coordinates of the mobile device.
in	<i>Bi_vec[ ]</i>	the measured magnetic field strength.
in, out	<i>f_vec[ ]</i>	errors related to reference stations and destined position.

Definition at line 29 of file magnetic\_based\_fi.c.

References `K_T`, and `matrix_t`.

Referenced by `magnetic_based_jacobian_get_JTf()`.

## 10.34 magnetic\_based\_fi.h File Reference

Error function of DC-pulsed, magnetic localization system.

```
#include <math.h>
#include <string.h>
#include "matrix.h"
```

### Functions

- void `magnetic_based_f_i` (uint8\_t ref\_points\_num, `matrix_t` ref\_points\_matrix[ ][3], `matrix_t` point[ ], `matrix_t` Bi\_vec[ ], `matrix_t` f\_vec[ ])
 

*Defines the error function of a magnetic-based localization system.*

### 10.34.1 Detailed Description

Error function of DC-pulsed, magnetic localization system.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

Naouar Guerchali

### 10.34.2 Function Documentation

#### 10.34.2.1 magnetic\_based\_f\_i()

```
void magnetic_based_f_i (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ ][3],
    matrix_t point[ ],
    matrix_t Bi_vec[ ],
    matrix_t f_vec[ ] )
```

Defines the error function of a magnetic-based localization system.

This error function is related to multiple reference stations.

## Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>Bi_vec</i> []	the measured magnetic field strength.
in, out	<i>f_vec</i> []	errors related to reference stations and destined position.

Definition at line 29 of file magnetic\_based\_fi.c.

References `K_T`, and `matrix_t`.

Referenced by `magnetic_based_jacobian_get_JTf()`.

## 10.35 magnetic\_based\_jacobian.c File Reference

Jacobian function of DC-pulsed, magnetic localization system.

```
#include <math.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "magnetic_based_fi.h"
#include "magnetic_based_position.h"
#include "matrix.h"
```

### Functions

- void `magnetic_based_jacobian_get_J` (uint8\_t ref\_point\_num, matrix\_t ref\_point\_matrix[ref\_point\_num][3], matrix\_t point[], matrix\_t J[ref\_point\_num][3])  
*Computes the Jacobian matrix of magnetic-based localization system.*
- void `magnetic_based_jacobian_get_JTJ` (uint8\_t ref\_points\_num, matrix\_t ref\_point\_matrix[ref\_points\_num][3], matrix\_t point[3], matrix\_t \*unused, matrix\_t JTJ[3][3])  
*Defines  $J_f^T J_f$  of magnetic-based localization system.*
- void `magnetic_based_jacobian_get_JTf` (uint8\_t ref\_points\_num, matrix\_t ref\_points\_matrix[ref\_points\_num][3], matrix\_t point[3], matrix\_t Bi\_vec[ref\_points\_num], matrix\_t JTf[3])  
*Defines  $J_f^T \vec{f}$  of magnetic-based localization system.*
- void `magnetic_based_jacobian_get_J_mul_s` (uint8\_t ref\_points\_num, matrix\_t ref\_point\_matrix[ref\_points\_num][3], matrix\_t point[3], matrix\_t s[3], matrix\_t J\_s[ref\_points\_num])  
*Computes  $J_f^T \vec{s}$  of magnetic-based localization system.*

### 10.35.1 Detailed Description

Jacobian function of DC-pulsed, magnetic localization system.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
Naouar Guerchali

## 10.35.2 Function Documentation

### 10.35.2.1 magnetic\_based\_jacobian\_get\_J()

```
void magnetic_based_jacobian_get_J (
    uint8_t ref_point_num,
    matrix_t ref_point_matrix[ref_point_num][3],
    matrix_t point[],
    matrix_t J[ref_point_num][3] )
```

Computes the Jacobian matrix of magnetic-based localization system.

#### Parameters

in	<i>ref_point_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in, out	<i>J</i> [][]	includes the Jacobian Matrix.

Definition at line 32 of file magnetic\_based\_jacobian.c.

References `K_T`, and `matrix_t`.

Referenced by `magnetic_based_jacobian_get_JTf()`, and `magnetic_based_jacobian_get_JTJ()`.

### 10.35.2.2 magnetic\_based\_jacobian\_get\_J\_mul\_s()

```
void magnetic_based_jacobian_get_J_mul_s (
    uint8_t ref_points_num,
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t s[3],
    matrix_t J_s[ref_points_num] )
```

Computes  $J_f^T \vec{s}$  of magnetic-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>s</i> []	correction vector.
in, out	<i>J_s</i> []	includes the $J_f^T \vec{s}$ vector.

Definition at line 130 of file magnetic\_based\_jacobian.c.

References K\_T, and matrix\_t.

### 10.35.2.3 magnetic\_based\_jacobian\_get\_JTf()

```
void magnetic_based_jacobian_get_JTf (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t Bi_vec[ref_points_num],
    matrix_t JTf[3] )
```

Defines  $J_f^T \vec{f}$  of magnetic-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>Bi_vec</i> []	magnetic field strengths of the coils (reference stations).
in, out	<i>JTf</i> []	includes the $J_f^T \vec{f}$ vector.

Definition at line 112 of file magnetic\_based\_jacobian.c.

References magnetic\_based\_f\_i(), magnetic\_based\_jacobian\_get\_J(), matrix\_t, and matrix\_trans\_mul\_vec().

### 10.35.2.4 magnetic\_based\_jacobian\_get\_JTJ()

```
void magnetic_based_jacobian_get_JTJ (
    uint8_t ref_points_num,
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t * unused,
    matrix_t JTJ[3][3] )
```

Defines  $J_f^T J_f$  of magnetic-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

## Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [[[]]]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>unused</i>	this variable can be set to NULL. It is introduced to guarantee the compatibility with the <a href="#">dist_based_jacobian_get_JTJ</a> , <a href="#">modified_gauss_newton</a> , and the <a href="#">opt_levenberg_marquardt</a> functions.
in, out	<i>JTJ</i> [[[]]]	includes the $J_f^T J_f$ matrix.

Definition at line 99 of file `magnetic_based_jacobian.c`.

References `magnetic_based_jacobian_get_J()`, `matrix_t`, and `matrix_trans_mul_itself()`.

## 10.36 magnetic\_based\_jacobian.h File Reference

Jacobian function of DC-pulsed, magnetic localization system.

```
#include "matrix.h"
```

### Functions

- void [magnetic\\_based\\_jacobian\\_get\\_J](#) (uint8\_t ref\_point\_num, [matrix\\_t](#) ref\_point\_matrix[ref\_point\_num][3], [matrix\\_t](#) point[], [matrix\\_t](#) J[ref\_point\_num][3])  
*Computes the Jacobian matrix of magnetic-based localization system.*
- void [magnetic\\_based\\_jacobian\\_get\\_JTJ](#) (uint8\_t ref\_points\_num, [matrix\\_t](#) ref\_point\_matrix[ref\_points\_num][3], [matrix\\_t](#) point[3], [matrix\\_t](#) \*unused, [matrix\\_t](#) JTJ[3][3])  
*Defines  $J_f^T J_f$  of magnetic-based localization system.*
- void [magnetic\\_based\\_jacobian\\_get\\_JTf](#) (uint8\_t ref\_points\_num, [matrix\\_t](#) ref\_points\_matrix[ref\_points\_num][3], [matrix\\_t](#) point[3], [matrix\\_t](#) Bi\_vec[ref\_points\_num], [matrix\\_t](#) JTf[3])  
*Defines  $J_f^T \vec{f}$  of magnetic-based localization system.*
- void [magnetic\\_based\\_jacobian\\_get\\_J\\_mul\\_s](#) (uint8\_t ref\_points\_num, [matrix\\_t](#) ref\_point\_matrix[ref\_points\_num][3], [matrix\\_t](#) point[3], [matrix\\_t](#) s[3], [matrix\\_t](#) J\_s[ref\_points\_num])  
*Computes  $J_f^T \vec{s}$  of magnetic-based localization system.*

### 10.36.1 Detailed Description

Jacobian function of DC-pulsed, magnetic localization system.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
Naouar Guerchali

### 10.36.2 Function Documentation

### 10.36.2.1 magnetic\_based\_jacobian\_get\_J()

```
void magnetic_based_jacobian_get_J (
    uint8_t ref_point_num,
    matrix_t ref_point_matrix[ref_point_num][3],
    matrix_t point[],
    matrix_t J[ref_point_num][3] )
```

Computes the Jacobian matrix of magnetic-based localization system.

#### Parameters

in	<i>ref_point_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in, out	<i>J</i> [][]	includes the Jacobian Matrix.

Definition at line 32 of file magnetic\_based\_jacobian.c.

References `K_T`, and `matrix_t`.

Referenced by `magnetic_based_jacobian_get_JTf()`, and `magnetic_based_jacobian_get_JTJ()`.

### 10.36.2.2 magnetic\_based\_jacobian\_get\_J\_mul\_s()

```
void magnetic_based_jacobian_get_J_mul_s (
    uint8_t ref_points_num,
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t s[3],
    matrix_t J_s[ref_points_num] )
```

Computes  $J_f^T \vec{s}$  of magnetic-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>s</i> []	correction vector.
in, out	<i>J_s</i> []	includes the $J_f^T \vec{s}$ vector.

Definition at line 130 of file magnetic\_based\_jacobian.c.

References `K_T`, and `matrix_t`.

### 10.36.2.3 magnetic\_based\_jacobian\_get\_JTf()

```
void magnetic_based_jacobian_get_JTf (
    uint8_t ref_points_num,
    matrix_t ref_points_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t Bi_vec[ref_points_num],
    matrix_t JTf[3] )
```

Defines  $J_f^T \vec{f}$  of magnetic-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_points_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>Bi_vec</i> []	magnetic field strengths of the coils (reference stations).
in, out	<i>JTf</i> []	includes the $J_f^T \vec{f}$ vector.

Definition at line 112 of file magnetic\_based\_jacobian.c.

References magnetic\_based\_f\_i(), magnetic\_based\_jacobian\_get\_J(), matrix\_t, and matrix\_trans\_mul\_vec().

### 10.36.2.4 magnetic\_based\_jacobian\_get\_JTJ()

```
void magnetic_based_jacobian_get_JTJ (
    uint8_t ref_points_num,
    matrix_t ref_point_matrix[ref_points_num][3],
    matrix_t point[3],
    matrix_t * unused,
    matrix_t JTJ[3][3] )
```

Defines  $J_f^T J_f$  of magnetic-based localization system.

Where  $J_f$  is the Jacobian matrix. This function is a part of derivatives to minimize the sum of square errors.

#### Parameters

in	<i>ref_points_num</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.
in	<i>unused</i>	this variable can be set to NULL. It is introduced to guarantee the compatibility with the <a href="#">dist_based_jacobian_get_JTJ</a> , <a href="#">modified_gauss_newton</a> , and the <a href="#">opt_levenberg_marquardt</a> functions.
in, out	<i>JTJ</i> [][]	includes the $J_f^T J_f$ matrix.

Definition at line 99 of file magnetic\_based\_jacobian.c.

References `magnetic_based_jacobian_get_J()`, `matrix_t`, and `matrix_trans_mul_itself()`.

## 10.37 magnetic\_based\_position.c File Reference

Functions of of DC-pulsed, magnetic localization system.

```
#include <math.h>
#include <complex.h>
#include <magnetic_based_position.h>
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include "matrix.h"
#include "vector.h"
#include "trilateration.h"
```

### Functions

- void `magnetic_based_get_absolute_error` (`matrix_t` value\_arr[], `matrix_t` approx\_value\_arr[], `matrix_t` absolute\_error\_arr[], `uint8_t` length)  
*Computes the absolute error of a position of magnet-based localization system.*
- `matrix_t` `magnetic_based_get_distances_to_anchors` (`matrix_t` ref\_point[3], `matrix_t` point[3])  
*Computes the distance between a mobile station and a reference station of magnetic-based localization system.*
- `matrix_t` `magnetic_based_get_magnetic_field` (`matrix_t` ref\_point[3], `matrix_t` target\_point[3], `matrix_t` k)  
*Computes the magnetic field strength from a mobile station to a reference station.*
- void `magnetic_based_get_magnetic_field_vec` (`uint8_t` m, `matrix_t` ref\_point\_matrix[m][3], `matrix_t` target\_point[], `matrix_t` k, `matrix_t` magn\_field\_vec[])  
*Computes the magnetic field strengths from a mobile station to various reference stations.*
- void `magnetic_based_get_distances` (`matrix_t` magnetic\_field\_strength\_arr[], `matrix_t` angular\_theta\_arr[], `matrix_t` distance\_arr[], `uint8_t` length, `matrix_t` k)  
*Computes the distances between a mobile station and the reference stations of a magnet-based localization system.*
- `matrix_t` `magnetic_based_get_r` (`matrix_t` B, `matrix_t` theta, `matrix_t` k)  
*Computes the distance between a mobile station and a reference stations of a magnet-based localization system.*
- void `magnetic_based_preprocessing_get_position` (`uint8_t` anchor\_num, `matrix_t` anchor\_pos\_matrix[anchor\_num][3], `matrix_t` pseudo\_inv\_matrix[4][anchor\_num], `matrix_t` homog\_sol\_arr[], `matrix_t` solution\_x1[], `matrix_t` solution\_x2[])  
*Computes the position of a mobile station by a magnetic-based localization system.*

### 10.37.1 Detailed Description

Functions of of DC-pulsed, magnetic localization system.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

## 10.37.2 Function Documentation

### 10.37.2.1 magnetic\_based\_get\_absolute\_error()

```
void magnetic_based_get_absolute_error (
    matrix_t value_arr[],
    matrix_t approx_value_arr[],
    matrix_t absolute_error_arr[],
    uint8_t length )
```

Computes the absolute error of a position of magnet-based localization system.

#### Parameters

in	<i>value_arr</i>	true position.
in	<i>approx_value_arr</i>	approximate position of the mobile device.
in, out	<i>absolute_error_arr</i>	includes the absolute error.
in	<i>length</i>	arrays length.

Definition at line 34 of file magnetic\_based\_position.c.

### 10.37.2.2 magnetic\_based\_get\_distances()

```
void magnetic_based_get_distances (
    matrix_t magnetic_field_strength_arr[],
    matrix_t angular_theta_arr[],
    matrix_t distance_arr[],
    uint8_t length,
    matrix_t k )
```

Computes the distances between a mobile station and the reference stations of a magnet-based localization system.

#### Parameters

in	<i>magnetic_field_strength_arr[]</i>	includes the magnetic field strengths.
in	<i>angular_theta_arr[]</i>	elevation angles.
in, out	<i>distance_arr[]</i>	distance array.
in	<i>length</i>	length of the arrays.
in	<i>k</i>	magnetic constant.

Definition at line 125 of file magnetic\_based\_position.c.

References `matrix_t`, and `MILPS_MAX_DIST`.

### 10.37.2.3 magnetic\_based\_get\_distances\_to\_anchors()

```
matrix_t magnetic_based_get_distances_to_anchors (
    matrix_t ref_point[3],
    matrix_t point[3] )
```

Computes the distance between a mobile station and a reference station of magnetic-based localization system.

#### Parameters

in	<i>ref_point</i> []	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of the mobile device.

#### Returns

the distance between the mobile station and the reference station.

Definition at line 50 of file magnetic\_based\_position.c.

References matrix\_t, vector\_get\_norm2(), and vector\_sub().

### 10.37.2.4 magnetic\_based\_get\_magnetic\_field()

```
matrix_t magnetic_based_get_magnetic_field (
    matrix_t ref_point[3],
    matrix_t target_point[3],
    matrix_t k )
```

Computes the magnetic field strength from a mobile station to a reference station.

#### Parameters

in	<i>ref_point</i> []	three-dimensional coordinates of a reference station.
in	<i>target_point</i> []	three-dimensional coordinates of the mobile device.
in	<i>k</i>	magnetic constant.

#### Returns

the magnetic field strength.

Definition at line 62 of file magnetic\_based\_position.c.

References matrix\_t, and vector\_flex\_print().

### 10.37.2.5 magnetic\_based\_get\_magnetic\_field\_vec()

```
void magnetic_based_get_magnetic_field_vec (
    uint8_t m,
    matrix_t ref_point_matrix[m][3],
    matrix_t target_point[],
    matrix_t k,
    matrix_t magn_field_vec[] )
```

Computes the magnetic field strengths from a mobile station to various reference stations.

## Parameters

in	<i>m</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>target_point</i> []	three-dimensional coordinates of the mobile device.
in	<i>k</i>	magnetic constant.
in, out	<i>magn_field_vec</i> []	includes the magnetic field strengths.

Definition at line 91 of file magnetic\_based\_position.c.

References `matrix_t`.

## 10.37.2.6 magnetic\_based\_get\_r()

```
matrix_t magnetic_based_get_r (
    matrix_t B,
    matrix_t theta,
    matrix_t k )
```

Computes the distance between a mobile station and a reference stations of a magnet-based localization system.

## Parameters

in	<i>B</i>	magnetic field strength.
in	<i>theta</i>	elevation angle.
in	<i>k</i>	magnetic constant.

## Returns

the distance between the mobile station and the reference station.

Definition at line 166 of file magnetic\_based\_position.c.

References `matrix_t`, and `MILPS_MAX_DIST`.

Referenced by `magnetic_based_test()`.

## 10.37.2.7 magnetic\_based\_preprocessing\_get\_position()

```
void magnetic_based_preprocessing_get_position (
    uint8_t anchor_num,
    matrix_t anchor_pos_matrix[anchor_num][3],
    matrix_t pseudo_inv_matrix[4][anchor_num],
    matrix_t homog_sol_arr[],
    matrix_t solution_x1[],
    matrix_t solution_x2[] )
```

Computes the position of a mobile station by a magnetic-based localization system.

The position is computed based on a pre-processed pseudo-inverse matrix and the homogeneous solution in the case of three reference stations.

## Parameters

in	<i>anchor_num</i>	number of the reference stations.
in	<i>anchor_pos_matrix</i> [][]	coordinates of the reference stations.
in	<i>pseudo_inv_matrix</i> [[[]]	pointer to the pre-processed pseudo-inverse matrix.
in	<i>homog_sol_arr</i> []	pointer to the homogeneous solution.
in, out	<i>solution_x1</i> []	includes the first solution.
in, out	<i>solution_x2</i> []	includes the second solution.

Definition at line 188 of file magnetic\_based\_position.c.

References `matrix_t`, and `trilateration1()`.

## 10.38 magnetic\_based\_position.h File Reference

Functions of of DC-pulsed, magnetic localization system.

```
#include <inttypes.h>
#include <math.h>
#include "matrix.h"
```

### Macros

- `#define IO 15`  
*The current running through the coil.*
- `#define NW 140`  
*The number of turns of the wire.*
- `#define R0 0.25`  
*The radius of the coil.*
- `#define PI 3.14159265358979323846`  
*The pi constant.*
- `#define AR PI * R0 * R0`  
*The base area of the coil.*
- `#define MU0 4 * PI * 1E-7`  
*The permeability of free space,.*
- `#define K MU0 * NW * IO * AR / (4 * PI)`  
*The K constant.*
- `#define MG_TO_T 1E7`  
*From Mega Gauss to Tesla.*
- `#define K_T K * MG_TO_T`  
*The number of turns of the wire.*
- `#define MILPS_MAX_DIST 25`  
*The number of turns of the wire.*

## Functions

- void `magnetic_based_get_absolute_error` (`matrix_t` value\_arr[], `matrix_t` approx\_value\_arr[], `matrix_t` absolute\_error\_arr[], `uint8_t` length)  
*Computes the absolute error of a position of magnet-based localization system.*
- void `magnetic_based_get_distances` (`matrix_t` magnetic\_field\_strength\_arr[], `matrix_t` angular\_theta\_arr[], `matrix_t` distance\_arr[], `uint8_t` length, `matrix_t` k)  
*Computes the distances between a mobile station and the reference stations of a magnet-based localization system.*
- `matrix_t` `magnetic_based_get_r` (`matrix_t` B, `matrix_t` theta, `matrix_t` k)  
*Computes the distance between a mobile station and a reference stations of a magnet-based localization system.*
- `matrix_t` `magnetic_based_get_distances_to_anchors` (`matrix_t` ref\_point[3], `matrix_t` point[3])  
*Computes the distance between a mobile station and a reference station of magnetic-based localization system.*
- `matrix_t` `magnetic_based_get_magnetic_field` (`matrix_t` ref\_point[3], `matrix_t` target\_point[3], `matrix_t` k)  
*Computes the magnetic field strength from a mobile station to a reference station.*
- void `magnetic_based_get_magnetic_field_vec` (`uint8_t` m, `matrix_t` ref\_point\_matrix[m][3], `matrix_t` target\_point[], `matrix_t` k, `matrix_t` magn\_field\_vec[])  
*Computes the magnetic field strengths from a mobile station to various reference stations.*
- void `magnetic_based_preprocessing_get_position` (`uint8_t` anchor\_num, `matrix_t` anchor\_pos\_matrix[anchor\_num][3], `matrix_t` pseudo\_inv\_matrix[4][anchor\_num], `matrix_t` homog\_sol\_arr[], `matrix_t` solution\_x1[], `matrix_t` solution\_x2[])  
*Computes the position of a mobile station by a magnetic-based localization system.*

### 10.38.1 Detailed Description

Functions of of DC-pulsed, magnetic localization system.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
 Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

### 10.38.2 Function Documentation

#### 10.38.2.1 magnetic\_based\_get\_absolute\_error()

```
void magnetic_based_get_absolute_error (
    matrix_t value_arr[],
    matrix_t approx_value_arr[],
    matrix_t absolute_error_arr[],
    uint8_t length )
```

Computes the absolute error of a position of magnet-based localization system.

## Parameters

in	<i>value_arr</i>	true position.
in	<i>approx_value_arr</i>	approximate position of the mobile device.
in, out	<i>absolute_error_arr</i>	includes the absolute error.
in	<i>length</i>	arrays length.

Definition at line 34 of file magnetic\_based\_position.c.

### 10.38.2.2 magnetic\_based\_get\_distances()

```
void magnetic_based_get_distances (
    matrix_t magnetic_field_strength_arr[],
    matrix_t angular_theta_arr[],
    matrix_t distance_arr[],
    uint8_t length,
    matrix_t k )
```

Computes the distances between a mobile station and the reference stations of a magnet-based localization system.

## Parameters

in	<i>magnetic_field_strength_arr[]</i>	includes the magnetic field strengths.
in	<i>angular_theta_arr[]</i>	elevation angles.
in, out	<i>distance_arr[]</i>	distance array.
in	<i>length</i>	length of the arrays.
in	<i>k</i>	magnetic constant.

Definition at line 125 of file magnetic\_based\_position.c.

References matrix\_t, and MILPS\_MAX\_DIST.

### 10.38.2.3 magnetic\_based\_get\_distances\_to\_anchors()

```
matrix_t magnetic_based_get_distances_to_anchors (
    matrix_t ref_point[3],
    matrix_t point[3] )
```

Computes the distance between a mobile station and a reference station of magnetic-based localization system.

## Parameters

in	<i>ref_point[]</i>	three-dimensional coordinates of the reference stations.
in	<i>point[]</i>	three-dimensional coordinates of the mobile device.

**Returns**

the distance between the mobile station and the reference station.

Definition at line 50 of file magnetic\_based\_position.c.

References matrix\_t, vector\_get\_norm2(), and vector\_sub().

**10.38.2.4 magnetic\_based\_get\_magnetic\_field()**

```
matrix_t magnetic_based_get_magnetic_field (
    matrix_t ref_point[3],
    matrix_t target_point[3],
    matrix_t k )
```

Computes the magnetic field strength from a mobile station to a reference station.

**Parameters**

in	<i>ref_point</i> []	three-dimensional coordinates of a reference station.
in	<i>target_point</i> []	three-dimensional coordinates of the mobile device.
in	<i>k</i>	magnetic constant.

**Returns**

the magnetic field strength.

Definition at line 62 of file magnetic\_based\_position.c.

References matrix\_t, and vector\_flex\_print().

**10.38.2.5 magnetic\_based\_get\_magnetic\_field\_vec()**

```
void magnetic_based_get_magnetic_field_vec (
    uint8_t m,
    matrix_t ref_point_matrix[m][3],
    matrix_t target_point[],
    matrix_t k,
    matrix_t magn_field_vec[] )
```

Computes the magnetic field strengths from a mobile station to various reference stations.

**Parameters**

in	<i>m</i>	number of the reference stations.
in	<i>ref_point_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>target_point</i> []	three-dimensional coordinates of the mobile device.
in	<i>k</i>	magnetic constant.
in, out	<i>magn_field_vec</i> []	includes the magnetic field strengths.

Definition at line 91 of file `magnetic_based_position.c`.

References `matrix_t`.

**10.38.2.6 magnetic\_based\_get\_r()**

```
matrix_t magnetic_based_get_r (
    matrix_t B,
    matrix_t theta,
    matrix_t k )
```

Computes the distance between a mobile station and a reference stations of a magnet-based localization system.

**Parameters**

in	<i>B</i>	magnetic field strength.
in	<i>theta</i>	elevation angle.
in	<i>k</i>	magnetic constant.

**Returns**

the distance between the mobile station and the reference station.

Definition at line 166 of file `magnetic_based_position.c`.

References `matrix_t`, and `MILPS_MAX_DIST`.

Referenced by `magnetic_based_test()`.

**10.38.2.7 magnetic\_based\_preprocessing\_get\_position()**

```
void magnetic_based_preprocessing_get_position (
    uint8_t anchor_num,
    matrix_t anchor_pos_matrix[anchor_num][3],
    matrix_t pseudo_inv_matrix[4][anchor_num],
    matrix_t homog_sol_arr[],
    matrix_t solution_x1[],
    matrix_t solution_x2[] )
```

Computes the position of a mobile station by a magnetic-based localization system.

The position is computed based on a pre-processed pseudo-inverse matrix and the homogeneous solution in the case of three reference stations.

## Parameters

in	<i>anchor_num</i>	number of the reference stations.
in	<i>anchor_pos_matrix</i> [][]	coordinates of the reference stations.
in	<i>pseudo_inv_matrix</i> [[[]]	pointer to the pre-processed pseudo-inverse matrix.
in	<i>homog_sol_arr</i> []	pointer to the homogeneous solution.
in, out	<i>solution_x1</i> []	includes the first solution.
in, out	<i>solution_x2</i> []	includes the second solution.

Definition at line 188 of file magnetic\_based\_position.c.

References `matrix_t`, and `trilateration1()`.

## 10.39 magnetic\_based\_test.c File Reference

Examples of localization algorithms of magnetic-based positioning systems.

```
#include <stdio.h>
#include "matrix.h"
#include "vector.h"
#include "utils.h"
#include "trilateration.h"
#include "magnetic_based_position.h"
```

### Functions

- void [magnetic\\_based\\_test](#) (void)  
*Example of a magnetic-based localization system.*

#### 10.39.1 Detailed Description

Examples of localization algorithms of magnetic-based positioning systems.

Localization algorithms examples using artificially generated DC-pulsed, magnetic signals (see the [methods of magnetic-based localization systems](#)).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
 Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

## 10.40 magnetic\_based\_test.h File Reference

Examples of localization algorithms of magnetic-based positioning systems.

## Functions

- void [magnetic\\_based\\_test](#) (void)

*Example of a magnetic-based localization system.*

### 10.40.1 Detailed Description

Examples of localization algorithms of magnetic-based positioning systems.

Localization algorithms examples using artificially generated DC-pulsed, magnetic signals (see the [methods of magnetic-based localization systems](#)).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

## 10.41 matrix.c File Reference

Matrix computations.

```
#include <stdint.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include <float.h>
#include "matrix.h"
#include "utils.h"
#include "svd.h"
```

## Functions

- void [matrix\\_clear](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n])  
*Clear all the elements of the vector.*
- void [matrix\\_init](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n], [matrix\\_t](#) value)  
*Initialize all the elements of the matrix with a specified value.*
- void [matrix\\_transpose](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) src\_matrix[m][n], [matrix\\_t](#) dest\_matrix[n][m])  
*Computes the transpose of a matrix.*
- void [matrix\\_in\\_place\\_transpose](#) (uint8\_t m, [matrix\\_t](#) matrix[ ][m])  
*Computes the in-place transpose of a matrix.*
- void [matrix\\_copy](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) src\_matrix[m][n], [matrix\\_t](#) dest\_matrix[m][n])  
*Copy the elements of a matrix to another matrix.*
- void [matrix\\_part\\_copy](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) src\_matrix[m][n], uint8\_t start\_row\_ind, uint8\_t end\_row\_ind, uint8\_t start\_col\_ind, uint8\_t end\_col\_ind, uint8\_t dest\_row\_num, uint8\_t dest\_col\_num, [matrix\\_t](#) dest\_matrix[ ][dest\_col\_num])  
*Copy a part of a matrix to another matrix or sub-matrix.*
- void [matrix\\_print](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n])  
*Display the values of the matrix elements.*

- void `matrix_part_print` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], uint8\_t start\_row\_ind, uint8\_t end\_row\_ind, uint8\_t start\_col\_ind, uint8\_t end\_col\_ind)  
*Display the values of the sub-matrix elements.*
- void `matrix_flex_print` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], uint8\_t before\_dot, uint8\_t after\_dot)  
*Display the values of the matrix elements.*
- void `matrix_flex_part_print` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], uint8\_t start\_row\_ind, uint8\_t end\_row\_ind, uint8\_t start\_col\_ind, uint8\_t end\_col\_ind, uint8\_t before\_dot, uint8\_t after\_dot)  
*Display the values of the sub-matrix elements.*
- uint8\_t `matrix_get_rank` (uint8\_t m, uint8\_t n, `matrix_t` singl\_values\_arr[], uint8\_t length)  
*Compute the rank of a matrix.*
- void `matrix_sub` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], `matrix_t` B[m][n], `matrix_t` A\_minus\_B[m][n])  
*Compute the subtraction of two matrices.*
- void `matrix_add` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], `matrix_t` B[m][n], `matrix_t` A\_plus\_B[m][n])  
*Compute the addition of two matrices.*
- void `matrix_add_to_diag` (uint8\_t n, `matrix_t` A[][n], uint8\_t diag\_el\_num, `matrix_t` value)  
*Add a number to diagonal elements of a matrix.*
- void `matrix_mul` (uint8\_t a\_line\_num, uint8\_t a\_col\_num, `matrix_t` a\_matrix[a\_line\_num][a\_col\_num], uint8\_t b\_line\_num, uint8\_t b\_col\_num, `matrix_t` b\_matrix[b\_line\_num][b\_col\_num], `matrix_t` dest\_matrix[a\_line\_num][b\_col\_num])  
*Compute the multiplication of two matrices.*
- void `matrix_part_mul` (uint8\_t a\_col\_num\_max, `matrix_t` a\_matrix[][a\_col\_num\_max], uint8\_t b\_col\_num\_max, `matrix_t` b\_matrix[][b\_col\_num\_max], uint8\_t a\_start\_row\_ind, uint8\_t a\_end\_row\_ind, uint8\_t a\_start\_col\_ind, uint8\_t a\_end\_col\_ind, uint8\_t b\_start\_row\_ind, uint8\_t b\_end\_row\_ind, uint8\_t b\_start\_col\_ind, uint8\_t b\_end\_col\_ind, uint8\_t dest\_col\_size, `matrix_t` dest\_matrix[][dest\_col\_size])  
*Compute the partial multiplication of two matrices.*
- void `matrix_mul_vec` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], `matrix_t` vec[n], `matrix_t` dst\_arr[m])  
*Compute the multiplication of a matrix with a column vector.*
- void `matrix_vec_mul_matr` (uint8\_t m, uint8\_t n, `matrix_t` vec[m], `matrix_t` matrix[m][n], `matrix_t` dst\_arr[n])  
*Compute the multiplication of row vector and a matrix.*
- void `matrix_mul_scalar_vec_matr` (uint8\_t m, uint8\_t n, `matrix_t` scalar, `matrix_t` vec[m], `matrix_t` matrix[m][n], `matrix_t` dst\_arr[n])  
*Compute the multiplication of a scalar with row vector and a matrix.*
- void `matrix_part_mul_scalar_vec_matr` (uint8\_t max\_m, uint8\_t max\_n, `matrix_t` scalar, `matrix_t` vec[max\_m], `matrix_t` matrix[max\_m][max\_n], uint8\_t begin\_row, uint8\_t begin\_column, `matrix_t` dst\_arr[max\_n - begin\_row])  
*Compute the multiplication of a scalar with row vector and a sub-matrix.*
- void `matrix_trans_mul_vec` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], uint8\_t b\_size, `matrix_t` b\_vec[m], `matrix_t` c\_vec[n])  
*Compute the multiplication of transposed matrix with column vector.*
- void `matrix_mul_col_vec_row_vec` (uint8\_t m, `matrix_t` col\_vec[m], uint8\_t n, `matrix_t` row\_vec[n], uint8\_t max\_n, `matrix_t` res\_mat[][max\_n])  
*Compute the multiplication of a column and row vector.*
- void `matrix_trans_mul_itself` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], `matrix_t` AT\_mul\_A[n][n])  
*Compute the multiplication of the transpose of a matrix with itself.*
- void `matrix_set_diag_elements` (uint8\_t m, uint8\_t n, `matrix_t` value, `matrix_t` diag\_matrix[m][n])  
*Set all the diagonal elements of a matrix with a specified value.*
- void `matrix_get_diag_mat_new` (uint8\_t m, uint8\_t n, `matrix_t` diag\_matrix[m][n], uint8\_t length, `matrix_t` vec[])  
*Set all the diagonal elements of a matrix with values that are saved in a vector.*
- void `matrix_get_diag_mat` (uint8\_t m, uint8\_t n, `matrix_t` value, `matrix_t` diag\_matrix[m][n])  
*Create a diagonal matrix with a specified value.*
- void `matrix_mul_scalar` (uint8\_t m, uint8\_t n, `matrix_t` mat\_src[m][n], `matrix_t` value, `matrix_t` mat\_dest[m][n])

*Multiply all elements of a matrix with a specified value.*

- void `matrix_get_column_vec` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], uint8\_t col\_num, `matrix_t` col\_vec[m])

*Get a column of a matrix.*

- void `matrix_get_part_column_vec` (uint8\_t max\_m, uint8\_t max\_n, `matrix_t` matrix[max\_m][max\_n], uint8\_t col\_num, uint8\_t offset, `matrix_t` col\_vec[max\_m - offset])

*Get a part of a column of a matrix.*

- `matrix_t` `matrix_get_max_elem_in_column` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], uint8\_t col\_num)

*Get the largest element of a column vector in a matrix.*

- `matrix_t` `matrix_get_abs_max_elem_in_column` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], uint8\_t col\_num)

*Get the maximum absolute value of a column vector in a matrix.*

- `matrix_t` `matrix_get_max_elem_in_part_column` (uint8\_t max\_m, uint8\_t max\_n, `matrix_t` matrix[max\_m][max\_n], uint8\_t row\_num, uint8\_t col\_num)

*Get the largest element of a column vector in a sub-matrix.*

- `matrix_t` `matrix_get_abs_max_elem_in_part_column` (uint8\_t max\_m, uint8\_t max\_n, `matrix_t` matrix[max\_m][max\_n], uint8\_t row\_num, uint8\_t col\_num)

*Get the maximum absolute value of a column vector in a sub-matrix.*

- `matrix_t` `matrix_get_abs_max_elem_and_index_in_part_column` (uint8\_t max\_m, uint8\_t max\_n, `matrix_t` matrix[max\_m][max\_n], uint8\_t row\_num, uint8\_t col\_num, uint8\_t \*index)

*Get the maximum absolute value and its position in a column vector in a sub-matrix.*

- void `matrix_swap_rows` (uint8\_t n, `matrix_t` matrix[][n], uint8\_t i, uint8\_t j)

*Swaps two rows of a matrix.*

- void `matrix_part_swap_rows` (uint8\_t n, `matrix_t` matrix[][n], uint8\_t i, uint8\_t j, uint8\_t col\_begin, uint8\_t col\_end)

*Swaps two rows of a sub-matrix.*

- double `matrix_get_two_norm` (uint8\_t m, uint8\_t n, `matrix_t` A[][n])

*Get the 2-norm of a matrix that is equal to the largest singular value.*

- double `matrix_get_frob_norm` (uint8\_t m, uint8\_t n, `matrix_t` A[][n])

*Get the Frobenius norm of a matrix.*

- void `matrix_get_inv_upper_triangular` (uint8\_t m, uint8\_t n, `matrix_t` U[][n], `matrix_t` inv\_U[][m])

*Computes the inverse an upper triangular matrix.*

- void `matrix_get_inv_lower_triangular` (uint8\_t m, uint8\_t n, `matrix_t` L[][n], `matrix_t` inv\_L[][m])

*Computes the inverse a lower triangular matrix.*

- void `matrix_get_upper_triangular` (uint8\_t m, uint8\_t n, `matrix_t` A[][n], `matrix_t` tr\_up\_A[][n])

*Gets the upper triangular part of a matrix.*

- void `matrix_get_lower_triangular` (uint8\_t m, uint8\_t n, `matrix_t` A[][n], `matrix_t` tr\_low\_A[][n])

*Gets the lower triangular part of a matrix.*

- `matrix_t` `matrix_read` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], uint8\_t i, uint8\_t j)

*Get the value of a matrix at the position (i,j).*

- void `matrix_write` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], uint8\_t i, uint8\_t j, `matrix_t` val)

*Write a value in a matrix at the position (i,j).*

### 10.41.1 Detailed Description

Matrix computations.

Matrix computations include operations such as addition, subtraction, and transposition.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.41.2 Function Documentation

### 10.41.2.1 matrix\_add()

```
void matrix_add (
    uint8_t m,
    uint8_t n,
    matrix_t A[m][n],
    matrix_t B[m][n],
    matrix_t A_plus_B[m][n] )
```

Compute the addition of two matrices.

Add matrix B to matrix A and return the result in A\_plus\_B matrix.

#### Parameters

in	<i>m</i>	row number of the matrix to add.
in	<i>n</i>	column number of the matrix to add.
in	<i>A</i> [][]	pointer to the first matrix.
in	<i>B</i> [][]	pointer to the second matrix.
out	<i>A_plus_B</i> [][]	pointer to the destination matrix.

Definition at line 341 of file matrix.c.

Referenced by matrix\_test().

### 10.41.2.2 matrix\_add\_to\_diag()

```
void matrix_add_to_diag (
    uint8_t n,
    matrix_t A[][n],
    uint8_t diag_el_num,
    matrix_t value )
```

Add a number to diagonal elements of a matrix.

#### Parameters

in	<i>n</i>	column number of the matrix.
in, out	<i>A</i> [][]	pointer to the matrix.
in	<i>diag_el_num</i>	number of diagonal elements to overwrite.
in	<i>value</i>	the value to add to the diagonal elements.

Definition at line 353 of file matrix.c.

Referenced by `opt_levenberg_marquardt()`, and `opt_levenberg_marquardt_correction()`.

### 10.41.2.3 `matrix_clear()`

```
void matrix_clear (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n] )
```

Clear all the elements of the vector.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
out	<i>matrix[ ][ ]</i>	pointer to the matrix.

Definition at line 46 of file `matrix.c`.

Referenced by `dist_based_jacobian_get_JTJ()`, `householder_test()`, `matrix_get_diag_mat()`, `matrix_get_diag_mat_new()`, `matrix_get_inv_low_triang()`, `matrix_get_inv_upp_triang()`, and `qr_givens_decomp()`.

### 10.41.2.4 `matrix_copy()`

```
void matrix_copy (
    uint8_t m,
    uint8_t n,
    matrix_t src_matrix[m][n],
    matrix_t dest_matrix[m][n] )
```

Copy the elements of a matrix to another matrix.

#### Parameters

in	<i>m</i>	row number of the matrix to copy.
in	<i>n</i>	column number of the matrix to copy.
in	<i>src_matrix[ ][ ]</i>	pointer to the source matrix
out	<i>dest_matrix[ ][ ]</i>	pointer to the destination matrix.

Definition at line 83 of file `matrix.c`.

References `matrix_t`.

Referenced by `givens_test()`, `householder_test()`, `matrix_get_two_norm()`, `solve_big_matrix_test()`, `solve_test()`, and `triangular_matrices_test()`.

### 10.41.2.5 matrix\_flex\_part\_print()

```
void matrix_flex_part_print (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t start_row_ind,
    uint8_t end_row_ind,
    uint8_t start_col_ind,
    uint8_t end_col_ind,
    uint8_t before_dot,
    uint8_t after_dot )
```

Display the values of the sub-matrix elements.

This function allows the user to determine the precision as well as the with of the numbers to display.

#### Note

This function is more memory-consuming than [matrix\\_part\\_print](#).

#### Parameters

in	<i>m</i>	total row number of the matrix.
in	<i>n</i>	total column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the entire matrix.
in	<i>start_row_ind</i>	start row number of the sub-matrix.
in	<i>end_row_ind</i>	end row number of the sub-matrix.
in	<i>start_col_ind</i>	start column number of the sub-matrix.
in	<i>end_col_ind</i>	end column number of the sub-matrix.
in	<i>before_dot</i>	the number of digits to be printed before the decimal point.
in	<i>after_dot</i>	the number of digits to be printed after the decimal point.

Definition at line 247 of file matrix.c.

References [utils\\_printf\(\)](#).

Referenced by [matrix\\_test\(\)](#).

### 10.41.2.6 matrix\_flex\_print()

```
void matrix_flex_print (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t before_dec,
    uint8_t after_dec )
```

Display the values of the matrix elements.

This function allows the user to determine the precision as well as the with of the numbers to display.

#### Note

This function is more memory-consuming than [matrix\\_print](#).

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the entire matrix.
in	<i>before_dec</i>	the number of digits to be printed before the decimal point.
in	<i>after_dec</i>	the number of digits to be printed after the decimal point.

Definition at line 220 of file matrix.c.

References `utils_printf()`.

Referenced by `givens_test()`, `householder_test()`, `inv_triangular_matrices_test()`, `lu_decomp_test()`, `matrix_test()`, `moore_penrose_pinv_compute_print()`, `optimization_test()`, `pos_algos_common_test()`, `solve_big_matrix_test()`, `solve_test()`, and `triangular_matrices_test()`.

#### 10.41.2.7 `matrix_get_abs_max_elem_and_index_in_part_column()`

```
matrix_t matrix_get_abs_max_elem_and_index_in_part_column (
    uint8_t max_m,
    uint8_t max_n,
    matrix_t matrix[max_m][max_n],
    uint8_t row_num,
    uint8_t col_num,
    uint8_t * index )
```

Get the maximum absolute value and its position in a column vector in a sub-matrix.

## Parameters

in	<i>max_m</i>	total row number of the matrix.
in	<i>max_n</i>	total column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the entire matrix.
in	<i>row_num</i>	row number of the sub-matrix.
in	<i>col_num</i>	column number of the sub-matrix.
out	<i>index</i>	pointer to the variable holding the position of the maximum absolute value in the column vector of the sub-matrix.

## Returns

the maximum absolute value of a partial column.

Definition at line 703 of file matrix.c.

References `matrix_t`.

Referenced by `lu_decomp()`.

**10.41.2.8 matrix\_get\_abs\_max\_elem\_in\_column()**

```
matrix_t matrix_get_abs_max_elem_in_column (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t col_num )
```

Get the maximum absolute value of a column vector in a matrix.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>col_num</i>	column number.

## Returns

the maximum absolute value of a column.

Definition at line 651 of file matrix.c.

References `matrix_t`.

### 10.41.2.9 matrix\_get\_abs\_max\_elem\_in\_part\_column()

```
matrix_t matrix_get_abs_max_elem_in_part_column (
    uint8_t max_m,
    uint8_t max_n,
    matrix_t matrix[max_m][max_n],
    uint8_t row_num,
    uint8_t col_num )
```

Get the maximum absolute value of a column vector in a sub-matrix.

#### Parameters

in	<i>max_m</i>	total row number of the matrix.
in	<i>max_n</i>	total column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the entire matrix.
in	<i>row_num</i>	row number of the sub-matrix.
in	<i>col_num</i>	column number of the sub-matrix.

#### Returns

the maximum absolute value of a partial column.

Definition at line 685 of file matrix.c.

References `matrix_t`.

### 10.41.2.10 matrix\_get\_column\_vec()

```
void matrix_get_column_vec (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t col_num,
    matrix_t col_vec[m] )
```

Get a column of a matrix.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>col_num</i>	number of the requested column.
out	<i>col_vec[ ]</i>	pointer to the column vector.

Definition at line 612 of file matrix.c.

Referenced by `matrix_trans_mul_itself()`.

### 10.41.2.11 matrix\_get\_diag\_mat()

```
void matrix_get_diag_mat (
    uint8_t m,
    uint8_t n,
    matrix_t value,
    matrix_t diag_matrix[m][n] )
```

Create a diagonal matrix with a specified value.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>value</i>	value of the diagonal elements.
in, out	<i>diag_matrix</i> [][]	pointer to the diagonal matrix.

Definition at line 594 of file matrix.c.

References `matrix_clear()`, and `matrix_set_diag_elements()`.

Referenced by `lu_decomp()`, and `matrix_test()`.

### 10.41.2.12 matrix\_get\_diag\_mat\_new()

```
void matrix_get_diag_mat_new (
    uint8_t m,
    uint8_t n,
    matrix_t diag_matrix[m][n],
    uint8_t length,
    matrix_t vec[] )
```

Set all the diagonal elements of a matrix with values that are saved in a vector.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in, out	<i>diag_matrix</i> [][]	pointer to the matrix.
in	<i>length</i>	size of the vector.
in	<i>vec</i>	pointer to the vector containing diagonal elements.

Definition at line 581 of file matrix.c.

References `matrix_clear()`.

Referenced by `matrix_test()`.

**10.41.2.13 matrix\_get\_frob\_norm()**

```
double matrix_get_frob_norm (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n] )
```

Get the Frobenius norm of a matrix.

**Parameters**

in	$m$	row number of the matrix.
in	$n$	column number of the matrix.
in	$A[ ][ ]$	pointer to the matrix.

**Returns**

the Frobenius norm a matrix.

Definition at line 782 of file matrix.c.

Referenced by matrix\_test().

**10.41.2.14 matrix\_get\_inv\_low\_triang()**

```
void matrix_get_inv_low_triang (
    uint8_t m,
    uint8_t n,
    matrix_t L[ ][n],
    matrix_t inv_L[ ][m] )
```

Computes the inverse a lower triangular matrix.

**Parameters**

in	$m$	row number of the matrix.
in	$n$	column number of the matrix.
in	$L[ ][ ]$	pointer to the matrix.
out	$inv\_L[ ][ ]$	pointer to the inverse matrix.

Definition at line 817 of file matrix.c.

References matrix\_clear(), and matrix\_t.

Referenced by inv\_triangular\_matrices\_test(), and solve\_lu\_decomp().

**10.41.2.15 matrix\_get\_inv\_upp\_triang()**

```
void matrix_get_inv_upp_triang (
    uint8_t m,
    uint8_t n,
    matrix_t U[ ][n],
    matrix_t inv_U[ ][m] )
```

Computes the inverse an upper triangular matrix.

**Parameters**

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>U[ ][ ]</i>	pointer to the upper triangular matrix.
out	<i>inv_U[ ][ ]</i>	pointer to the inverse matrix.

Definition at line 795 of file matrix.c.

References `matrix_clear()`, and `matrix_t`.

Referenced by `inv_triangular_matrices_test()`.

**10.41.2.16 matrix\_get\_low\_triang()**

```
void matrix_get_low_triang (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t tr_low_A[ ][n] )
```

Gets the lower triangular part of a matrix.

**Parameters**

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>A[ ][ ]</i>	pointer to the matrix.
out	<i>tr_low_A[ ][ ]</i>	pointer to the lower triangular part of the matrix.

Definition at line 868 of file matrix.c.

Referenced by `triangular_matrices_test()`.

**10.41.2.17 matrix\_get\_max\_elem\_in\_column()**

```
matrix_t matrix_get_max_elem_in_column (
    uint8_t m,
```

```
uint8_t n,
matrix_t matrix[m][n],
uint8_t col_num )
```

Get the largest element of a column vector in a matrix.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>col_num</i>	column number.

#### Returns

the largest element of a column.

Definition at line 635 of file matrix.c.

References `matrix_t`.

#### 10.41.2.18 matrix\_get\_max\_elem\_in\_part\_column()

```
matrix_t matrix_get_max_elem_in_part_column (
    uint8_t max_m,
    uint8_t max_n,
    matrix_t matrix[max_m][max_n],
    uint8_t row_num,
    uint8_t col_num )
```

Get the largest element of a column vector in a sub-matrix.

#### Parameters

in	<i>max_m</i>	total row number of the matrix.
in	<i>max_n</i>	total column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the entire matrix.
in	<i>row_num</i>	row number of the sub-matrix.
in	<i>col_num</i>	column number of the sub-matrix.

#### Returns

the largest element of a partial column.

Definition at line 668 of file matrix.c.

References `matrix_t`.

**10.41.2.19 matrix\_get\_part\_column\_vec()**

```
void matrix_get_part_column_vec (
    uint8_t max_m,
    uint8_t max_n,
    matrix_t matrix[max_m][max_n],
    uint8_t col_num,
    uint8_t offset,
    matrix_t col_vec[max_m - offset] )
```

Get a part of a column of a matrix.

**Parameters**

in	<i>max_m</i>	row number of the matrix.
in	<i>max_n</i>	column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>col_num</i>	number of the requested column.
in	<i>offset</i>	points to the start position of the column vector.
out	<i>col_vec[ ]</i>	pointer to the column vector.

Definition at line 623 of file matrix.c.

**10.41.2.20 matrix\_get\_rank()**

```
uint8_t matrix_get_rank (
    uint8_t m,
    uint8_t n,
    matrix_t singl_values_arr[ ],
    uint8_t length )
```

Compute the rank of a matrix.

The SVD must be previously invoked to get the singular values of the matrix.

**Note**

This function should be invoked after the call of the [svd](#) method.

**Parameters**

in	<i>m</i>	row number of the source matrix.
in	<i>n</i>	column number of the source matrix.
in	<i>singl_values_arr[ ]</i>	array containing the singular values of the matrix.
in	<i>length</i>	length of the singular values array.

**Returns**

the rank of the matrix.

Definition at line 312 of file matrix.c.

Referenced by `trilateration_get_rank_and_homogeneous_solution()`.

**10.41.2.21 matrix\_get\_two\_norm()**

```
double matrix_get_two_norm (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n] )
```

Get the 2-norm of a matrix that is equal to the largest singular value.

**Parameters**

in	$m$	row number of the matrix.
in	$n$	column number of the matrix.
in	$A[ ][ ]$	pointer to the matrix.

**Returns**

the 2-norm of a matrix.

Definition at line 752 of file matrix.c.

References `matrix_dim_t::col_num`, `matrix_copy()`, `matrix_print()`, `matrix_t`, `matrix_dim_t::row_num`, `svd()`, `svd_↔get_single_values_num()`, and `svd_get_U_dim()`.

Referenced by `matrix_test()`.

**10.41.2.22 matrix\_get\_upp\_triang()**

```
void matrix_get_upp_triang (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t tr_up_A[ ][n] )
```

Gets the upper triangular part of a matrix.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>A</i> [][]	pointer to the matrix.
out	<i>tr_up_A</i> [][]	pointer to the upper triangular part of the matrix.

Definition at line 841 of file matrix.c.

Referenced by `qr_givens_decomp()`, and `triangular_matrices_test()`.

### 10.41.2.23 matrix\_in\_place\_transpose()

```
void matrix_in_place_transpose (
    uint8_t m,
    matrix_t matrix[][m] )
```

Computes the in-place transpose of a matrix.

Transpose the matrix without auxiliary memory.

## Note

This function is limited to square matrices.

## Parameters

in	<i>m</i>	row and column number of the matrix.
in, out	<i>matrix</i> [][]	pointer to the matrix to transpose.

Definition at line 69 of file matrix.c.

References `matrix_t`.

### 10.41.2.24 matrix\_init()

```
void matrix_init (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    matrix_t value )
```

Initialize all the elements of the matrix with a specified value.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
out	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>value</i>	value to be set.

Definition at line 51 of file matrix.c.

#### 10.41.2.25 matrix\_mul()

```
void matrix_mul (
    uint8_t a_line_num,
    uint8_t a_col_num,
    matrix_t a_matrix[a_line_num][a_col_num],
    uint8_t b_line_num,
    uint8_t b_col_num,
    matrix_t b_matrix[b_line_num][b_col_num],
    matrix_t dest_matrix[a_line_num][b_col_num] )
```

Compute the multiplication of two matrices.

## Parameters

in	<i>a_line_num</i>	row number of the first matrix.
in	<i>a_col_num</i>	column number of the first matrix.
in	<i>a_matrix[ ][ ]</i>	pointer to the first matrix.
in	<i>b_line_num</i>	row number of the second matrix.
in	<i>b_col_num</i>	column number of the second matrix.
in	<i>b_matrix[ ][ ]</i>	pointer to the second matrix.
out	<i>dest_matrix[ ][ ]</i>	pointer to the destination matrix.

Definition at line 363 of file matrix.c.

Referenced by get\_PDOP(), matrix\_test(), and solve\_lu\_decomp().

#### 10.41.2.26 matrix\_mul\_col\_vec\_row\_vec()

```
void matrix_mul_col_vec_row_vec (
    uint8_t m,
    matrix_t col_vec[m],
    uint8_t n,
    matrix_t row_vec[n],
    uint8_t max_n,
    matrix_t res_mat[ ][max_n] )
```

Compute the multiplication of a column and row vector.

Return  $C_{m,1} * R_{1,n} = M_{m,n}$ , where C is a m-dimensional column vector, R is a n-dimensional row vector, and the result is a (mxn)-matrix.

## Parameters

in	<i>m</i>	row number of the column vector.
in	<i>col_vec[]</i>	pointer to the column vector.
in	<i>n</i>	column number of the row vector.
in	<i>row_vec[]</i>	pointer to the row vector.
in	<i>max_n</i>	column number of the result matrix.
out	<i>res_mat[][]</i>	pointer to the (mxn) result matrix.

Definition at line 531 of file matrix.c.

## 10.41.2.27 matrix\_mul\_scalar()

```
void matrix_mul_scalar (
    uint8_t m,
    uint8_t n,
    matrix_t mat_src[m][n],
    matrix_t value,
    matrix_t mat_dest[m][n] )
```

Multiply all elements of a matrix with a specified value.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>mat_src</i>	pointer to the source matrix.
in	<i>value</i>	multiplication factor.
out	<i>mat_dest[][]</i>	pointer to the destination matrix.

Definition at line 602 of file matrix.c.

Referenced by loc\_levenberg\_marquardt(), loc\_levenberg\_marquardt\_correction(), opt\_levenberg\_marquardt(), and opt\_levenberg\_marquardt\_correction().

## 10.41.2.28 matrix\_mul\_scalar\_vec\_matr()

```
void matrix_mul_scalar_vec_matr (
    uint8_t m,
    uint8_t n,
    matrix_t scalar,
    matrix_t vec[m],
    matrix_t matrix[m][n],
    matrix_t dst_arr[n] )
```

Compute the multiplication of a scalar with row vector and a matrix.

Return  $scal * R_{1,m} * A_{m,n} = R_{1,n}$ , where *scal* is a scalar, *R* is a *m*-dimensional row vector, *A* is a (*m*×*n*)-matrix, and the result is a row vector of *n*-dimension.

**Parameters**

in	<i>m</i>	size of the row vector and row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>scalar</i>	scalar value.
in	<i>vec[]</i>	pointer to the row vector.
in	<i>matrix[][]</i>	pointer to the matrix.
out	<i>dst_arr</i>	pointer to the destination row vector of n-dimension.

Definition at line 470 of file matrix.c.

References `matrix_t`.

**10.41.2.29 matrix\_mul\_vec()**

```
void matrix_mul_vec (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    matrix_t vec[n],
    matrix_t dst_arr[m] )
```

Compute the multiplication of a matrix with a column vector.

Return  $A_{m,n} * V_{n,1} = B_{m,1}$ , where the A is a (mxn)-matrix, V is a n-dimensional column vector, and the result is a m-dimensional column vector.

**Parameters**

in	<i>m</i>	row number of the matrix to multiply.
in	<i>n</i>	column number of the matrix to multiply.
in	<i>matrix[][]</i>	pointer to the matrix.
in	<i>vec</i>	pointer to the n-dimensional column vector.
out	<i>dst_arr</i>	pointer to the destination m-dimensional column vector.

Definition at line 434 of file matrix.c.

References `matrix_t`.

Referenced by `get_delta_x()`, `loc_gauss_newton()`, `modified_gauss_newton()`, `newton_raphson()`, `opt_levenberg_↵_marquardt_correction()`, `solve()`, `solve_lu_decomp()`, `solve_test()`, and `trilateration_get_particular_solution()`.

**10.41.2.30 matrix\_part\_copy()**

```
void matrix_part_copy (
    uint8_t m,
```

```

uint8_t n,
matrix_t src_matrix[m][n],
uint8_t start_row_ind,
uint8_t end_row_ind,
uint8_t start_col_ind,
uint8_t end_col_ind,
uint8_t dest_row_num,
uint8_t dest_col_num,
matrix_t dest_matrix[][dest_col_num] )

```

Copy a part of a matrix to another matrix or sub-matrix.

A part of the source matrix can be copied in a sub-part of the destination matrix (sub-matrix). The source and destination sub-matrices are limited by the row and column indices.

#### Parameters

in	<i>m</i>	row number of the source matrix.
in	<i>n</i>	column number of the source matrix.
in	<i>src_matrix</i> [][]	pointer to the source matrix.
in	<i>start_row_ind</i>	the start index of the rows of the source sub-matrix.
in	<i>end_row_ind</i>	the end index of the rows of the source sub-matrix.
in	<i>start_col_ind</i>	the start index of the columns of the source sub-matrix.
in	<i>end_col_ind</i>	the end index of the columns of the source sub-matrix.
in	<i>dest_row_num</i>	the row number of the destination sub-matrix.
in	<i>dest_col_num</i>	the column number of the destination sub-matrix.
out	<i>dest_matrix</i> [][]	pointer to the destination (sub)-matrix.

Definition at line 89 of file matrix.c.

Referenced by `qr_common_get_reduced_QR()`, and `qr_givens_decomp()`.

#### 10.41.2.31 matrix\_part\_mul()

```

void matrix_part_mul (
    uint8_t a_col_num_max,
    matrix_t a_matrix[][a_col_num_max],
    uint8_t b_col_num_max,
    matrix_t b_matrix[][b_col_num_max],
    uint8_t a_start_row_ind,
    uint8_t a_end_row_ind,
    uint8_t a_start_col_ind,
    uint8_t a_end_col_ind,
    uint8_t b_start_row_ind,
    uint8_t b_end_row_ind,
    uint8_t b_start_col_ind,
    uint8_t b_end_col_ind,
    uint8_t dest_col_size,
    matrix_t dest_matrix[][dest_col_size] )

```

Compute the partial multiplication of two matrices.

Enables the calculation of matrix product of parts of two matrices.

## Parameters

in	<i>a_col_num_max</i>	column number of the first matrix.
in	<i>a_matrix</i> [][]	pointer to the first matrix.
in	<i>b_col_num_max</i>	column number of the second matrix.
in	<i>b_matrix</i> [][]	pointer to the second matrix.
in	<i>a_start_row_ind</i>	row begin of the first, partial matrix.
in	<i>a_end_row_ind</i>	row end of the first, partial matrix.
in	<i>a_start_col_ind</i>	column begin of the first, partial matrix.
in	<i>a_end_col_ind</i>	column end of the first, partial matrix.
in	<i>b_start_row_ind</i>	row begin of the second, partial matrix.
in	<i>b_end_row_ind</i>	row end of the second, partial matrix.
in	<i>b_start_col_ind</i>	column begin of the second, partial matrix.
in	<i>b_end_col_ind</i>	column end of the second, partial matrix.
in	<i>dest_col_size</i>	column size of the destination matrix.
out	<i>dest_matrix</i> [][]	pointer to the destination matrix.

Definition at line 391 of file matrix.c.

Referenced by `matrix_test()`.

#### 10.41.2.32 `matrix_part_mul_scalar_vec_matr()`

```
void matrix_part_mul_scalar_vec_matr (
    uint8_t max_m,
    uint8_t max_n,
    matrix_t scalar,
    matrix_t vec[max_m],
    matrix_t matrix[max_m][max_n],
    uint8_t begin_row,
    uint8_t begin_column,
    matrix_t dst_arr[max_n - begin_row] )
```

Compute the multiplication of a scalar with row vector and a sub-matrix.

Return  $\text{scal} * R_{1,m} * A_{m,n} = R_{1,n}$ , where *scal* is a scalar, *R* is a *m*-dimensional row vector, *A* is a (*m*×*n*)-matrix, and the result is a row vector of *n*-dimension.

## Parameters

in	<i>max_m</i>	size of the row vector and row number of the matrix.
in	<i>max_n</i>	column number of the matrix.
in	<i>scalar</i>	scalar value.
in	<i>vec</i> []	pointer to the row vector.
in	<i>matrix</i> [][]	pointer to the matrix.
in	<i>begin_row</i>	start row number of the sub-matrix.
in	<i>begin_column</i>	start column number of the sub-matrix.
out	<i>dst_arr</i>	pointer to the destination row vector of <i>n</i> -dimension.

Definition at line 489 of file matrix.c.

References `matrix_t`.

#### 10.41.2.33 `matrix_part_print()`

```
void matrix_part_print (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t start_row_ind,
    uint8_t end_row_ind,
    uint8_t start_col_ind,
    uint8_t end_col_ind )
```

Display the values of the sub-matrix elements.

##### Parameters

in	<i>m</i>	total row number of the matrix.
in	<i>n</i>	total column number of the matrix.
in	<i>matrix[][]</i>	pointer to the entire matrix.
in	<i>start_row_ind</i>	start row number of the sub-matrix.
in	<i>end_row_ind</i>	end row number of the sub-matrix.
in	<i>start_col_ind</i>	start column number of the sub-matrix.
in	<i>end_col_ind</i>	end column number of the sub-matrix.

Definition at line 164 of file matrix.c.

#### 10.41.2.34 `matrix_part_swap_rows()`

```
void matrix_part_swap_rows (
    uint8_t n,
    matrix_t matrix[][n],
    uint8_t i,
    uint8_t j,
    uint8_t col_begin,
    uint8_t col_end )
```

Swaps two rows of a sub-matrix.

Swaps the rows *i* and *j* of a part of a matrix.

**Parameters**

in	<i>n</i>	column number of the entire matrix.
in, out	<i>matrix</i> [][]	pointer to the entire matrix.
in	<i>i</i>	the i-the row of the sub-matrix.
in	<i>j</i>	the j-the row of the sub-matrix.
in	<i>col_begin</i>	the column begin of the sub-matrix.
in	<i>col_end</i>	the column end of the sub-matrix.

Definition at line 736 of file matrix.c.

References `matrix_t`.

Referenced by `lu_decomp()`.

**10.41.2.35 matrix\_print()**

```
void matrix_print (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n] )
```

Display the values of the matrix elements.

**Parameters**

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix</i> [][]	pointer to the entire matrix.

Definition at line 141 of file matrix.c.

Referenced by `matrix_get_two_norm()`, and `svd_compute_print_U_S_V_s()`.

**10.41.2.36 matrix\_read()**

```
matrix_t matrix_read (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t i,
    uint8_t j )
```

Get the value of a matrix at the position (i,j).

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix</i> [][]	pointer to the matrix.
in	<i>i</i>	row number.
in	<i>j</i>	column number.

## Returns

the value of the matrix at the row *i* and column *j*.

Definition at line 895 of file matrix.c.

## 10.41.2.37 matrix\_set\_diag\_elements()

```
void matrix_set_diag_elements (
    uint8_t m,
    uint8_t n,
    matrix_t value,
    matrix_t diag_matrix[m][n] )
```

Set all the diagonal elements of a matrix with a specified value.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>value</i>	value to be set.
in, out	<i>diag_matrix</i> [][]	pointer to the matrix.

Definition at line 565 of file matrix.c.

Referenced by matrix\_get\_diag\_mat().

## 10.41.2.38 matrix\_sub()

```
void matrix_sub (
    uint8_t m,
    uint8_t n,
    matrix_t A[m][n],
    matrix_t B[m][n],
    matrix_t A_minus_B[m][n] )
```

Compute the subtraction of two matrices.

Subtract matrix B from matrix A and return the result in A\_minus\_B matrix.

**Parameters**

in	$m$	row number of the matrix to add.
in	$n$	column number of the matrix to add.
in	$A[[]]$	pointer to the first matrix.
in	$B[[]]$	pointer to the second matrix.
out	$A\_minus\_B[[]]$	pointer to the destination matrix.

Definition at line 329 of file matrix.c.

Referenced by matrix\_test().

**10.41.2.39 matrix\_swap\_rows()**

```
void matrix_swap_rows (
    uint8_t n,
    matrix_t matrix[] [n],
    uint8_t i,
    uint8_t j )
```

Swaps two rows of a matrix.

Swaps the rows  $i$  and  $j$  of a matrix.

**Parameters**

in	$n$	column number of the matrix.
in	$matrix[[]]$	pointer to the matrix.
in	$i$	the $i$ -the row of the matrix.
in	$j$	the $j$ -the row of the matrix.

Definition at line 725 of file matrix.c.

References matrix\_t.

Referenced by lu\_decomp(), and matrix\_test().

**10.41.2.40 matrix\_trans\_mul\_itself()**

```
void matrix_trans_mul_itself (
    uint8_t m,
    uint8_t n,
    matrix_t A[m] [n],
    matrix_t AT_mul_A[n] [n] )
```

Compute the multiplication of the transpose of a matrix with itself.

Transpose the matrix  $A$  and multiply it with the matrix  $A$ :  $A' * A$ .

## Parameters

in	$m$	row number of the matrix.
in	$n$	column number of the matrix.
in	$A[[]]$	pointer to the matrix.
out	$AT\_mul\_A[[]]$	pointer to the destination matrix ( $A^t * A$ ).

Definition at line 545 of file matrix.c.

References `matrix_get_column_vec()`, and `matrix_t`.

Referenced by `magnetic_based_jacobian_get_JTJ()`, `modified_gauss_newton()`, `opt_levenberg_marquardt()`, and `opt_levenberg_marquardt_correction()`.

10.41.2.41 `matrix_trans_mul_vec()`

```
void matrix_trans_mul_vec (
    uint8_t m,
    uint8_t n,
    matrix_t A[m][n],
    uint8_t b_size,
    matrix_t b_vec[m],
    matrix_t c_vec[n] )
```

Compute the multiplication of transposed matrix with column vector.

Transpose A and return  $A^t n, m * V_{m,1} = B_{n,1}$ , where A is a (mxn)-matrix, V is a m-dimensional column vector, and the result is a n-dimensional column vector.

## Parameters

in	$m$	row number of the matrix to transpose and multiply.
in	$n$	column number of the matrix to transpose and multiply.
in	$A[[]]$	pointer to the matrix.
in	$b\_size$	size of the column vector.
in	$b\_vec[[]]$	pointer to the column vector of m-dimension.
out	$c\_vec[[]]$	pointer to the destination, column vector of n-dimension.

Definition at line 511 of file matrix.c.

Referenced by `magnetic_based_jacobian_get_JTf()`, `modified_gauss_newton()`, `opt_levenberg_marquardt()`, `opt_levenberg_marquardt_correction()`, `solve_givens()`, and `solve_householder()`.

10.41.2.42 `matrix_transpose()`

```
void matrix_transpose (
    uint8_t m,
```

```

uint8_t n,
matrix_t src_matrix[m][n],
matrix_t dest_matrix[n][m] )

```

Computes the transpose of a matrix.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>src_matrix</i> [][]	pointer to the matrix to transpose.
out	<i>dest_matrix</i> [][]	pointer to the destination matrix.

Definition at line 56 of file matrix.c.

Referenced by `get_PDOP()`, `matrix_test()`, `moore_penrose_get_pinv()`, and `triangular_matrices_test()`.

#### 10.41.2.43 matrix\_vec\_mul\_matr()

```

void matrix_vec_mul_matr (
    uint8_t m,
    uint8_t n,
    matrix_t vec[m],
    matrix_t matrix[m][n],
    matrix_t dst_arr[n] )

```

Compute the multiplication of row vector and a matrix.

Return  $R_{1,m} * A_{m,n} = R_{1,n}$ , where  $R$  is a  $m$ -dimensional row vector,  $A$  is a  $(m \times n)$ -matrix, and the result is a row vector of  $n$ -dimension.

#### Parameters

in	<i>m</i>	size of the row vector and row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>vec</i> []	pointer to the row vector.
in	<i>matrix</i> [][]	pointer to the matrix.
out	<i>dst_arr</i>	pointer to the destination row vector of $n$ -dimension.

Definition at line 452 of file matrix.c.

References `matrix_t`.

#### 10.41.2.44 matrix\_write()

```

void matrix_write (
    uint8_t m,

```

```

uint8_t n,
matrix_t matrix[m][n],
uint8_t i,
uint8_t j,
matrix_t val )

```

Write a value in a matrix at the position (i,j).

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
out	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>i</i>	row number.
in	<i>j</i>	column number.
in	<i>val</i>	value to write in the matrix.

Definition at line 901 of file matrix.c.

## 10.42 matrix.h File Reference

Matrix computations.

```

#include <inttypes.h>
#include <stdbool.h>

```

### Data Structures

- struct [matrix\\_dim\\_t](#)  
*A structure to define the row and column number of a matrix.*

### Macros

- #define [matrix\\_t](#) double  
*Define the data type of the matrix elements.*
- #define [MACHEPS](#) 2E-16
- #define [M\\_PI](#) 3.14159265358979323846

### Functions

- void [matrix\\_init](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n], [matrix\\_t](#) value)  
*Initialize all the elements of the matrix with a specified value.*
- void [matrix\\_clear](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n])  
*Clear all the elements of the vector.*
- void [matrix\\_transpose](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) src\_matrix[m][n], [matrix\\_t](#) dest\_matrix[n][m])  
*Computes the transpose of a matrix.*
- void [matrix\\_in\\_place\\_transpose](#) (uint8\_t m, [matrix\\_t](#) matrix[ ][m])

*Computes the in-place transpose of a matrix.*

- void `matrix_copy` (uint8\_t m, uint8\_t n, `matrix_t` src\_matrix[m][n], `matrix_t` dest\_matrix[m][n])

*Copy the elements of a matrix to another matrix.*

- void `matrix_part_copy` (uint8\_t m, uint8\_t n, `matrix_t` src\_matrix[m][n], uint8\_t start\_row\_ind, uint8\_t end\_row\_ind, uint8\_t start\_col\_ind, uint8\_t end\_col\_ind, uint8\_t dest\_row\_num, uint8\_t dest\_col\_num, `matrix_t` dest\_matrix[][dest\_col\_num])

*Copy a part of a matrix to another matrix or sub-matrix.*

- uint8\_t `matrix_get_rank` (uint8\_t m, uint8\_t n, `matrix_t` singl\_values\_arr[], uint8\_t length)

*Compute the rank of a matrix.*

- void `matrix_add` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], `matrix_t` B[m][n], `matrix_t` A\_plus\_B[m][n])

*Compute the addition of two matrices.*

- void `matrix_add_to_diag` (uint8\_t n, `matrix_t` A[][n], uint8\_t diag\_el\_num, `matrix_t` value)

*Add a number to diagonal elements of a matrix.*

- void `matrix_sub` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], `matrix_t` B[m][n], `matrix_t` A\_minus\_B[m][n])

*Compute the subtraction of two matrices.*

- void `matrix_mul` (uint8\_t a\_line\_num, uint8\_t a\_col\_num, `matrix_t` a\_matrix[a\_line\_num][a\_col\_num], uint8\_t b\_line\_num, uint8\_t b\_col\_num, `matrix_t` b\_matrix[b\_line\_num][b\_col\_num], `matrix_t` dest\_matrix[a\_line\_num][b\_col\_num])

*Compute the multiplication of two matrices.*

- void `matrix_part_mul` (uint8\_t a\_col\_num\_max, `matrix_t` a\_matrix[][a\_col\_num\_max], uint8\_t b\_col\_num\_max, `matrix_t` b\_matrix[][b\_col\_num\_max], uint8\_t a\_start\_row\_ind, uint8\_t a\_end\_row\_ind, uint8\_t a\_start\_col\_ind, uint8\_t a\_end\_col\_ind, uint8\_t b\_start\_row\_ind, uint8\_t b\_end\_row\_ind, uint8\_t b\_start\_col\_ind, uint8\_t b\_end\_col\_ind, uint8\_t dest\_col\_size, `matrix_t` dest\_matrix[][dest\_col\_size])

*Compute the partial multiplication of two matrices.*

- void `matrix_mul_vec` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], `matrix_t` vec[n], `matrix_t` dst\_arr[m])

*Compute the multiplication of a matrix with a column vector.*

- void `matrix_trans_mul_vec` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], uint8\_t b\_size, `matrix_t` b\_vec[m], `matrix_t` c\_vec[n])

*Compute the multiplication of transposed matrix with column vector.*

- void `matrix_mul_col_vec_row_vec` (uint8\_t m, `matrix_t` col\_vec[m], uint8\_t n, `matrix_t` row\_vec[n], uint8\_t max\_n, `matrix_t` res\_mat[][max\_n])

*Compute the multiplication of a column and row vector.*

- void `matrix_vec_mul_matr` (uint8\_t m, uint8\_t n, `matrix_t` vec[m], `matrix_t` matrix[m][n], `matrix_t` dst\_arr[n])

*Compute the multiplication of row vector and a matrix.*

- void `matrix_mul_scalar_vec_matr` (uint8\_t m, uint8\_t n, `matrix_t` scalar, `matrix_t` vec[m], `matrix_t` matrix[m][n], `matrix_t` dst\_arr[n])

*Compute the multiplication of a scalar with row vector and a matrix.*

- void `matrix_trans_mul_itself` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], `matrix_t` AT\_mul\_A[n][n])

*Compute the multiplication of the transpose of a matrix with itself.*

- void `matrix_set_diag_elements` (uint8\_t m, uint8\_t n, `matrix_t` value, `matrix_t` diag\_matrix[m][n])

*Set all the diagonal elements of a matrix with a specified value.*

- void `matrix_get_diag_mat_new` (uint8\_t m, uint8\_t n, `matrix_t` diag\_matrix[m][n], uint8\_t length, `matrix_t` vec[])

*Set all the diagonal elements of a matrix with values that are saved in a vector.*

- void `matrix_get_diag_mat` (uint8\_t m, uint8\_t n, `matrix_t` value, `matrix_t` diag\_matrix[m][n])

*Create a diagonal matrix with a specified value.*

- void `matrix_mul_scalar` (uint8\_t m, uint8\_t n, `matrix_t` mat\_src[m][n], `matrix_t` value, `matrix_t` mat\_dest[m][n])

*Multiply all elements of a matrix with a specified value.*

- void `matrix_get_column_vec` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], uint8\_t col\_num, `matrix_t` col\_vec[m])

*Get a column of a matrix.*

- void `matrix_get_part_column_vec` (uint8\_t max\_m, uint8\_t max\_n, `matrix_t` matrix[max\_m][max\_n], uint8\_t col\_num, uint8\_t offset, `matrix_t` col\_vec[max\_m - offset])

*Get a part of a column of a matrix.*

- [matrix\\_t matrix\\_get\\_max\\_elem\\_in\\_column](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n], uint8\_t col\_num)

*Get the largest element of a column vector in a matrix.*

- [matrix\\_t matrix\\_get\\_abs\\_max\\_elem\\_in\\_column](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n], uint8\_t col\_num)

*Get the maximum absolute value of a column vector in a matrix.*

- [matrix\\_t matrix\\_get\\_max\\_elem\\_in\\_part\\_column](#) (uint8\_t max\_m, uint8\_t max\_n, [matrix\\_t](#) matrix[max\_m][max\_n], uint8\_t row\_num, uint8\_t col\_num)

*Get the largest element of a column vector in a sub-matrix.*

- [matrix\\_t matrix\\_get\\_abs\\_max\\_elem\\_in\\_part\\_column](#) (uint8\_t max\_m, uint8\_t max\_n, [matrix\\_t](#) matrix[max\_m][max\_n], uint8\_t row\_num, uint8\_t col\_num)

*Get the maximum absolute value of a column vector in a sub-matrix.*

- [matrix\\_t matrix\\_get\\_abs\\_max\\_elem\\_and\\_index\\_in\\_part\\_column](#) (uint8\_t max\_m, uint8\_t max\_n, [matrix\\_t](#) matrix[max\_m][max\_n], uint8\_t row\_num, uint8\_t col\_num, uint8\_t \*index)

*Get the maximum absolute value and its position in a column vector in a sub-matrix.*

- void [matrix\\_swap\\_rows](#) (uint8\_t n, [matrix\\_t](#) matrix[][n], uint8\_t i, uint8\_t j)

*Swaps two rows of a matrix.*

- void [matrix\\_part\\_swap\\_rows](#) (uint8\_t n, [matrix\\_t](#) matrix[][n], uint8\_t i, uint8\_t j, uint8\_t col\_begin, uint8\_t col\_end)

*Swaps two rows of a sub-matrix.*

- double [matrix\\_get\\_two\\_norm](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) A[][n])

*Get the 2-norm of a matrix that is equal to the largest singular value.*

- double [matrix\\_get\\_frob\\_norm](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) A[][n])

*Get the Frobenius norm of a matrix.*

- void [matrix\\_get\\_inv\\_upper\\_triangular](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) U[][n], [matrix\\_t](#) inv\_U[][n])

*Computes the inverse an upper triangular matrix.*

- void [matrix\\_get\\_inv\\_lower\\_triangular](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) L[][n], [matrix\\_t](#) inv\_L[][n])

*Computes the inverse a lower triangular matrix.*

- void [matrix\\_get\\_upper\\_triangular](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) A[][n], [matrix\\_t](#) tr\_upper\_A[][n])

*Gets the upper triangular part of a matrix.*

- void [matrix\\_get\\_lower\\_triangular](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) A[][n], [matrix\\_t](#) tr\_lower\_A[][n])

*Gets the lower triangular part of a matrix.*

- void [matrix\\_print](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n])

*Display the values of the matrix elements.*

- void [matrix\\_part\\_print](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n], uint8\_t start\_row\_ind, uint8\_t end\_row\_ind, uint8\_t start\_col\_ind, uint8\_t end\_col\_ind)

*Display the values of the sub-matrix elements.*

- void [matrix\\_flex\\_print](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n], uint8\_t before\_dec, uint8\_t after\_dec)

*Display the values of the matrix elements.*

- void [matrix\\_flex\\_part\\_print](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n], uint8\_t start\_row\_ind, uint8\_t end\_row\_ind, uint8\_t start\_col\_ind, uint8\_t end\_col\_ind, uint8\_t before\_dot, uint8\_t after\_dot)

*Display the values of the sub-matrix elements.*

- void [matrix\\_part\\_mul\\_scalar\\_vec\\_matr](#) (uint8\_t max\_m, uint8\_t max\_n, [matrix\\_t](#) scalar, [matrix\\_t](#) vec[max\_m], [matrix\\_t](#) matrix[max\_m][max\_n], uint8\_t begin\_row, uint8\_t begin\_column, [matrix\\_t](#) dst\_arr[max\_n - begin\_row])

*Compute the multiplication of a scalar with row vector and a sub-matrix.*

- [matrix\\_t matrix\\_read](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n], uint8\_t i, uint8\_t j)

*Get the value of a matrix at the position (i,j).*

- void [matrix\\_write](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) matrix[m][n], uint8\_t i, uint8\_t j, [matrix\\_t](#) val)

*Write a value in a matrix at the position (i,j).*

### 10.42.1 Detailed Description

Matrix computations.

Matrix computations include operations such as addition, subtraction, and transposition.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.42.2 Macro Definition Documentation

#### 10.42.2.1 M\_PI

```
#define M_PI 3.14159265358979323846
```

Pi, the ratio of a circle's circumference to its diameter.

Definition at line 54 of file matrix.h.

#### 10.42.2.2 MACHEPS

```
#define MACHEPS 2E-16
```

The upper bound on the relative error due to rounding in floating point arithmetic.

Definition at line 46 of file matrix.h.

#### 10.42.2.3 matrix\_t

```
#define matrix_t double
```

Define the data type of the matrix elements.

The user can choose between a double or floating point arithmetic.

Definition at line 38 of file matrix.h.

### 10.42.3 Function Documentation

#### 10.42.3.1 matrix\_add()

```
void matrix_add (
    uint8_t m,
    uint8_t n,
    matrix_t A[m][n],
    matrix_t B[m][n],
    matrix_t A_plus_B[m][n] )
```

Compute the addition of two matrices.

Add matrix B to matrix A and return the result in A\_plus\_B matrix.

## Parameters

in	<i>m</i>	row number of the matrix to add.
in	<i>n</i>	column number of the matrix to add.
in	<i>A</i> [][]	pointer to the first matrix.
in	<i>B</i> [][]	pointer to the second matrix.
out	<i>A_plus_B</i> [][]	pointer to the destination matrix.

Definition at line 341 of file matrix.c.

Referenced by `matrix_test()`.

### 10.42.3.2 matrix\_add\_to\_diag()

```
void matrix_add_to_diag (
    uint8_t n,
    matrix_t A[ ][n],
    uint8_t diag_el_num,
    matrix_t value )
```

Add a number to diagonal elements of a matrix.

## Parameters

in	<i>n</i>	column number of the matrix.
in, out	<i>A</i> [][]	pointer to the matrix.
in	<i>diag_el_num</i>	number of diagonal elements to overwrite.
in	<i>value</i>	the value to add to the diagonal elements.

Definition at line 353 of file matrix.c.

Referenced by `opt_levenberg_marquardt()`, and `opt_levenberg_marquardt_correction()`.

### 10.42.3.3 matrix\_clear()

```
void matrix_clear (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n] )
```

Clear all the elements of the vector.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
out	<i>matrix</i> [][]	pointer to the matrix.

Definition at line 46 of file matrix.c.

Referenced by `dist_based_jacobian_get_JTJ()`, `householder_test()`, `matrix_get_diag_mat()`, `matrix_get_diag_mat_new()`, `matrix_get_inv_low_triang()`, `matrix_get_inv_upp_triang()`, and `qr_givens_decomp()`.

#### 10.42.3.4 `matrix_copy()`

```
void matrix_copy (
    uint8_t m,
    uint8_t n,
    matrix_t src_matrix[m][n],
    matrix_t dest_matrix[m][n] )
```

Copy the elements of a matrix to another matrix.

##### Parameters

in	<i>m</i>	row number of the matrix to copy.
in	<i>n</i>	column number of the matrix to copy.
in	<i>src_matrix</i> [][]	pointer to the source matrix
out	<i>dest_matrix</i> [][]	pointer to the destination matrix.

Definition at line 83 of file matrix.c.

References `matrix_t`.

Referenced by `givens_test()`, `householder_test()`, `matrix_get_two_norm()`, `solve_big_matrix_test()`, `solve_test()`, and `triangular_matrices_test()`.

#### 10.42.3.5 `matrix_flex_part_print()`

```
void matrix_flex_part_print (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t start_row_ind,
    uint8_t end_row_ind,
    uint8_t start_col_ind,
    uint8_t end_col_ind,
    uint8_t before_dot,
    uint8_t after_dot )
```

Display the values of the sub-matrix elements.

This function allows the user to determine the precision as well as the with of the numbers to display.

##### Note

This function is more memory-consuming than `matrix_part_print`.

## Parameters

in	<i>m</i>	total row number of the matrix.
in	<i>n</i>	total column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the entire matrix.
in	<i>start_row_ind</i>	start row number of the sub-matrix.
in	<i>end_row_ind</i>	end row number of the sub-matrix.
in	<i>start_col_ind</i>	start column number of the sub-matrix.
in	<i>end_col_ind</i>	end column number of the sub-matrix.
in	<i>before_dot</i>	the number of digits to be printed before the decimal point.
in	<i>after_dot</i>	the number of digits to be printed after the decimal point.

Definition at line 247 of file matrix.c.

References `utils_printf()`.

Referenced by `matrix_test()`.

10.42.3.6 `matrix_flex_print()`

```
void matrix_flex_print (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t before_dec,
    uint8_t after_dec )
```

Display the values of the matrix elements.

This function allows the user to determine the precision as well as the with of the numbers to display.

## Note

This function is more memory-consuming than `matrix_print`.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the entire matrix.
in	<i>before_dec</i>	the number of digits to be printed before the decimal point.
in	<i>after_dec</i>	the number of digits to be printed after the decimal point.

Definition at line 220 of file matrix.c.

References `utils_printf()`.

Referenced by `givens_test()`, `householder_test()`, `inv_triangular_matrices_test()`, `lu_decomp_test()`, `matrix_test()`, `moore_penrose_pinv_compute_print()`, `optimization_test()`, `pos_algos_common_test()`, `solve_big_matrix_test()`, `solve_test()`, and `triangular_matrices_test()`.

### 10.42.3.7 matrix\_get\_abs\_max\_elem\_and\_index\_in\_part\_column()

```
matrix_t matrix_get_abs_max_elem_and_index_in_part_column (
    uint8_t max_m,
    uint8_t max_n,
    matrix_t matrix[max_m][max_n],
    uint8_t row_num,
    uint8_t col_num,
    uint8_t * index )
```

Get the maximum absolute value and its position in a column vector in a sub-matrix.

#### Parameters

in	<i>max_m</i>	total row number of the matrix.
in	<i>max_n</i>	total column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the entire matrix.
in	<i>row_num</i>	row number of the sub-matrix.
in	<i>col_num</i>	column number of the sub-matrix.
out	<i>index</i>	pointer to the variable holding the position of the maximum absolute value in the column vector of the sub-matrix.

#### Returns

the maximum absolute value of a partial column.

Definition at line 703 of file matrix.c.

References matrix\_t.

Referenced by lu\_decomp().

### 10.42.3.8 matrix\_get\_abs\_max\_elem\_in\_column()

```
matrix_t matrix_get_abs_max_elem_in_column (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t col_num )
```

Get the maximum absolute value of a column vector in a matrix.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>col_num</i>	column number.

**Returns**

the maximum absolute value of a column.

Definition at line 651 of file matrix.c.

References `matrix_t`.

**10.42.3.9 matrix\_get\_abs\_max\_elem\_in\_part\_column()**

```
matrix_t matrix_get_abs_max_elem_in_part_column (
    uint8_t max_m,
    uint8_t max_n,
    matrix_t matrix[max_m][max_n],
    uint8_t row_num,
    uint8_t col_num )
```

Get the maximum absolute value of a column vector in a sub-matrix.

**Parameters**

in	<i>max_m</i>	total row number of the matrix.
in	<i>max_n</i>	total column number of the matrix.
in	<i>matrix[][]</i>	pointer to the entire matrix.
in	<i>row_num</i>	row number of the sub-matrix.
in	<i>col_num</i>	column number of the sub-matrix.

**Returns**

the maximum absolute value of a partial column.

Definition at line 685 of file matrix.c.

References `matrix_t`.

**10.42.3.10 matrix\_get\_column\_vec()**

```
void matrix_get_column_vec (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t col_num,
    matrix_t col_vec[m] )
```

Get a column of a matrix.

**Parameters**

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>col_num</i>	number of the requested column.
out	<i>col_vec[ ]</i>	pointer to the column vector.

Definition at line 612 of file matrix.c.

Referenced by `matrix_trans_mul_itself()`.

**10.42.3.11 matrix\_get\_diag\_mat()**

```
void matrix_get_diag_mat (
    uint8_t m,
    uint8_t n,
    matrix_t value,
    matrix_t diag_matrix[m][n] )
```

Create a diagonal matrix with a specified value.

**Parameters**

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>value</i>	value of the diagonal elements.
in, out	<i>diag_matrix[ ][ ]</i>	pointer to the diagonal matrix.

Definition at line 594 of file matrix.c.

References `matrix_clear()`, and `matrix_set_diag_elements()`.

Referenced by `lu_decomp()`, and `matrix_test()`.

**10.42.3.12 matrix\_get\_diag\_mat\_new()**

```
void matrix_get_diag_mat_new (
    uint8_t m,
    uint8_t n,
    matrix_t diag_matrix[m][n],
    uint8_t length,
    matrix_t vec[ ] )
```

Set all the diagonal elements of a matrix with values that are saved in a vector.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in, out	<i>diag_matrix</i> [[ <i>i</i> ]]	pointer to the matrix.
in	<i>length</i>	size of the vector.
in	<i>vec</i>	pointer to the vector containing diagonal elements.

Definition at line 581 of file matrix.c.

References `matrix_clear()`.

Referenced by `matrix_test()`.

### 10.42.3.13 `matrix_get_frob_norm()`

```
double matrix_get_frob_norm (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n] )
```

Get the Frobenius norm of a matrix.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>A</i> [[ <i>i</i> ]]	pointer to the matrix.

## Returns

the Frobenius norm a matrix.

Definition at line 782 of file matrix.c.

Referenced by `matrix_test()`.

### 10.42.3.14 `matrix_get_inv_low_triang()`

```
void matrix_get_inv_low_triang (
    uint8_t m,
    uint8_t n,
    matrix_t L[ ][n],
    matrix_t inv_L[ ][m] )
```

Computes the inverse a lower triangular matrix.

**Parameters**

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>L</i> [][]	pointer to the matrix.
out	<i>inv_L</i> [][]	pointer to the inverse matrix.

Definition at line 817 of file matrix.c.

References `matrix_clear()`, and `matrix_t`.

Referenced by `inv_triangular_matrices_test()`, and `solve_lu_decomp()`.

**10.42.3.15 matrix\_get\_inv\_upp\_triang()**

```
void matrix_get_inv_upp_triang (
    uint8_t m,
    uint8_t n,
    matrix_t U[ ][n],
    matrix_t inv_U[ ][m] )
```

Computes the inverse an upper triangular matrix.

**Parameters**

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>U</i> [][]	pointer to the upper triangular matrix.
out	<i>inv_U</i> [][]	pointer to the inverse matrix.

Definition at line 795 of file matrix.c.

References `matrix_clear()`, and `matrix_t`.

Referenced by `inv_triangular_matrices_test()`.

**10.42.3.16 matrix\_get\_low\_triang()**

```
void matrix_get_low_triang (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t tr_low_A[ ][n] )
```

Gets the lower triangular part of a matrix.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>A</i> [][]	pointer to the matrix.
out	<i>tr_low_A</i> [][]	pointer to the lower triangular part of the matrix.

Definition at line 868 of file matrix.c.

Referenced by `triangular_matrices_test()`.

### 10.42.3.17 matrix\_get\_max\_elem\_in\_column()

```
matrix_t matrix_get_max_elem_in_column (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t col_num )
```

Get the largest element of a column vector in a matrix.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix</i> [][]	pointer to the matrix.
in	<i>col_num</i>	column number.

## Returns

the largest element of a column.

Definition at line 635 of file matrix.c.

References `matrix_t`.

### 10.42.3.18 matrix\_get\_max\_elem\_in\_part\_column()

```
matrix_t matrix_get_max_elem_in_part_column (
    uint8_t max_m,
    uint8_t max_n,
    matrix_t matrix[max_m][max_n],
    uint8_t row_num,
    uint8_t col_num )
```

Get the largest element of a column vector in a sub-matrix.

**Parameters**

in	<i>max_m</i>	total row number of the matrix.
in	<i>max_n</i>	total column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the entire matrix.
in	<i>row_num</i>	row number of the sub-matrix.
in	<i>col_num</i>	column number of the sub-matrix.

**Returns**

the largest element of a partial column.

Definition at line 668 of file matrix.c.

References `matrix_t`.

**10.42.3.19 matrix\_get\_part\_column\_vec()**

```
void matrix_get_part_column_vec (
    uint8_t max_m,
    uint8_t max_n,
    matrix_t matrix[max_m][max_n],
    uint8_t col_num,
    uint8_t offset,
    matrix_t col_vec[max_m - offset] )
```

Get a part of a column of a matrix.

**Parameters**

in	<i>max_m</i>	row number of the matrix.
in	<i>max_n</i>	column number of the matrix.
in	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>col_num</i>	number of the requested column.
in	<i>offset</i>	points to the start position of the column vector.
out	<i>col_vec[ ]</i>	pointer to the column vector.

Definition at line 623 of file matrix.c.

**10.42.3.20 matrix\_get\_rank()**

```
uint8_t matrix_get_rank (
    uint8_t m,
    uint8_t n,
    matrix_t singl_values_arr[],
    uint8_t length )
```

Compute the rank of a matrix.

The SVD must be previously invoked to get the singular values of the matrix.

#### Note

This function should be invoked after the call of the [svd](#) method.

#### Parameters

in	<i>m</i>	row number of the source matrix.
in	<i>n</i>	column number of the source matrix.
in	<i>singl_values_arr[]</i>	array containing the singular values of the matrix.
in	<i>length</i>	length of the singular values array.

#### Returns

the rank of the matrix.

Definition at line 312 of file matrix.c.

Referenced by `trilateration_get_rank_and_homogeneous_solution()`.

#### 10.42.3.21 matrix\_get\_two\_norm()

```
double matrix_get_two_norm (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n] )
```

Get the 2-norm of a matrix that is equal to the largest singular value.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>A[ ][ ]</i>	pointer to the matrix.

#### Returns

the 2-norm of a matrix.

Definition at line 752 of file matrix.c.

References `matrix_dim_t::col_num`, `matrix_copy()`, `matrix_print()`, `matrix_t`, `matrix_dim_t::row_num`, `svd()`, `svd_↵  
get_single_values_num()`, and `svd_get_U_dim()`.

Referenced by `matrix_test()`.

### 10.42.3.22 matrix\_get\_upp\_triang()

```
void matrix_get_upp_triang (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t tr_up_A[ ][n] )
```

Gets the upper triangular part of a matrix.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>A[ ][ ]</i>	pointer to the matrix.
out	<i>tr_up_A[ ][ ]</i>	pointer to the upper triangular part of the matrix.

Definition at line 841 of file matrix.c.

Referenced by qr\_givens\_decomp(), and triangular\_matrices\_test().

### 10.42.3.23 matrix\_in\_place\_transpose()

```
void matrix_in_place_transpose (
    uint8_t m,
    matrix_t matrix[ ][m] )
```

Computes the in-place transpose of a matrix.

Transpose the matrix without auxiliary memory.

#### Note

This function is limited to square matrices.

#### Parameters

in	<i>m</i>	row and column number of the matrix.
in, out	<i>matrix[ ][ ]</i>	pointer to the matrix to transpose.

Definition at line 69 of file matrix.c.

References matrix\_t.

### 10.42.3.24 matrix\_init()

```
void matrix_init (
    uint8_t m,
```

```
uint8_t n,  
matrix_t matrix[m][n],  
matrix_t value )
```

Initialize all the elements of the matrix with a specified value.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
out	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>value</i>	value to be set.

Definition at line 51 of file matrix.c.

#### 10.42.3.25 matrix\_mul()

```
void matrix_mul (   
    uint8_t a_line_num,  
    uint8_t a_col_num,  
    matrix_t a_matrix[a_line_num][a_col_num],  
    uint8_t b_line_num,  
    uint8_t b_col_num,  
    matrix_t b_matrix[b_line_num][b_col_num],  
    matrix_t dest_matrix[a_line_num][b_col_num] )
```

Compute the multiplication of two matrices.

## Parameters

in	<i>a_line_num</i>	row number of the first matrix.
in	<i>a_col_num</i>	column number of the first matrix.
in	<i>a_matrix[[]]</i>	pointer to the first matrix.
in	<i>b_line_num</i>	row number of the second matrix.
in	<i>b_col_num</i>	column number of the second matrix.
in	<i>b_matrix[[]]</i>	pointer to the second matrix.
out	<i>dest_matrix[[]]</i>	pointer to the destination matrix.

Definition at line 363 of file matrix.c.

Referenced by `get_PDOP()`, `matrix_test()`, and `solve_lu_decomp()`.

#### 10.42.3.26 `matrix_mul_col_vec_row_vec()`

```
void matrix_mul_col_vec_row_vec (
    uint8_t m,
    matrix_t col_vec[m],
    uint8_t n,
    matrix_t row_vec[n],
    uint8_t max_n,
    matrix_t res_mat[ ][max_n] )
```

Compute the multiplication of a column and row vector.

Return  $C_{m,1} * R_{1,n} = M_{m,n}$ , where  $C$  is a  $m$ -dimensional column vector,  $R$  is a  $n$ -dimensional row vector, and the result is a  $(mxn)$ -matrix.

## Parameters

in	<i>m</i>	row number of the column vector.
in	<i>col_vec[ ]</i>	pointer to the column vector.
in	<i>n</i>	column number of the row vector.
in	<i>row_vec[ ]</i>	pointer to the row vector.
in	<i>max_n</i>	column number of the result matrix.
out	<i>res_mat[[]]</i>	pointer to the $(mxn)$ result matrix.

Definition at line 531 of file matrix.c.

#### 10.42.3.27 `matrix_mul_scalar()`

```
void matrix_mul_scalar (
    uint8_t m,
    uint8_t n,
    matrix_t mat_src[m][n],
```

```
matrix_t value,
matrix_t mat_dest[m][n] )
```

Multiply all elements of a matrix with a specified value.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>mat_src</i>	pointer to the source matrix.
in	<i>value</i>	multiplication factor.
out	<i>mat_dest[ ][ ]</i>	pointer to the destination matrix.

Definition at line 602 of file matrix.c.

Referenced by `loc_levenberg_marquardt()`, `loc_levenberg_marquardt_correction()`, `opt_levenberg_marquardt()`, and `opt_levenberg_marquardt_correction()`.

#### 10.42.3.28 matrix\_mul\_scalar\_vec\_matr()

```
void matrix_mul_scalar_vec_matr (
    uint8_t m,
    uint8_t n,
    matrix_t scalar,
    matrix_t vec[m],
    matrix_t matrix[m][n],
    matrix_t dst_arr[n] )
```

Compute the multiplication of a scalar with row vector and a matrix.

Return  $scal * R1, m * A_{m,n} = R1, n$ , where *scal* is a scalar, *R* is a *m*-dimensional row vector, *A* is a (*m*×*n*)-matrix, and the result is a row vector of *n*-dimension.

#### Parameters

in	<i>m</i>	size of the row vector and row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>scalar</i>	scalar value.
in	<i>vec[ ]</i>	pointer to the row vector.
in	<i>matrix[ ][ ]</i>	pointer to the matrix.
out	<i>dst_arr</i>	pointer to the destination row vector of <i>n</i> -dimension.

Definition at line 470 of file matrix.c.

References `matrix_t`.

### 10.42.3.29 matrix\_mul\_vec()

```
void matrix_mul_vec (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    matrix_t vec[n],
    matrix_t dst_arr[m] )
```

Compute the multiplication of a matrix with a column vector.

Return  $A_{m,n} * V_{n,1} = B_{m,1}$ , where the A is a (mxn)-matrix, V is a n-dimensional column vector, and the result is a m-dimensional column vector.

#### Parameters

in	<i>m</i>	row number of the matrix to multiply.
in	<i>n</i>	column number of the matrix to multiply.
in	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>vec</i>	pointer to the n-dimensional column vector.
out	<i>dst_arr</i>	pointer to the destination m-dimensional column vector.

Definition at line 434 of file matrix.c.

References matrix\_t.

Referenced by get\_delta\_x(), loc\_gauss\_newton(), modified\_gauss\_newton(), newton\_raphson(), opt\_levenberg↵\_marquardt\_correction(), solve(), solve\_lu\_decomp(), solve\_test(), and trilateration\_get\_particular\_solution().

### 10.42.3.30 matrix\_part\_copy()

```
void matrix_part_copy (
    uint8_t m,
    uint8_t n,
    matrix_t src_matrix[m][n],
    uint8_t start_row_ind,
    uint8_t end_row_ind,
    uint8_t start_col_ind,
    uint8_t end_col_ind,
    uint8_t dest_row_num,
    uint8_t dest_col_num,
    matrix_t dest_matrix[ ][dest_col_num] )
```

Copy a part of a matrix to another matrix or sub-matrix.

A part of the source matrix can be copied in a sub-part of the destination matrix (sub-matrix). The source and destination sub-matrices are limited by the row and column indices.

#### Parameters

in	<i>m</i>	row number of the source matrix.
in	<i>n</i>	column number of the source matrix.

## Parameters

in	<i>src_matrix</i> [][]	pointer to the source matrix.
in	<i>start_row_ind</i>	the start index of the rows of the source sub-matrix.
in	<i>end_row_ind</i>	the end index of the rows of the source sub-matrix.
in	<i>start_col_ind</i>	the start index of the columns of the source sub-matrix.
in	<i>end_col_ind</i>	the end index of the columns of the source sub-matrix.
in	<i>dest_row_num</i>	the row number of the destination sub-matrix.
in	<i>dest_col_num</i>	the column number of the destination sub-matrix.
out	<i>dest_matrix</i> [][]	pointer to the destination (sub)-matrix.

Definition at line 89 of file matrix.c.

Referenced by `qr_common_get_reduced_QR()`, and `qr_givens_decomp()`.

### 10.42.3.31 matrix\_part\_mul()

```
void matrix_part_mul (
    uint8_t a_col_num_max,
    matrix_t a_matrix[][a_col_num_max],
    uint8_t b_col_num_max,
    matrix_t b_matrix[][b_col_num_max],
    uint8_t a_start_row_ind,
    uint8_t a_end_row_ind,
    uint8_t a_start_col_ind,
    uint8_t a_end_col_ind,
    uint8_t b_start_row_ind,
    uint8_t b_end_row_ind,
    uint8_t b_start_col_ind,
    uint8_t b_end_col_ind,
    uint8_t dest_col_size,
    matrix_t dest_matrix[][dest_col_size] )
```

Compute the partial multiplication of two matrices.

Enables the calculation of matrix product of parts of two matrices.

## Parameters

in	<i>a_col_num_max</i>	column number of the first matrix.
in	<i>a_matrix</i> [][]	pointer to the first matrix.
in	<i>b_col_num_max</i>	column number of the second matrix.
in	<i>b_matrix</i> [][]	pointer to the second matrix.
in	<i>a_start_row_ind</i>	row begin of the first, partial matrix.
in	<i>a_end_row_ind</i>	row end of the first, partial matrix.
in	<i>a_start_col_ind</i>	column begin of the first, partial matrix.
in	<i>a_end_col_ind</i>	column end of the first, partial matrix.
in	<i>b_start_row_ind</i>	row begin of the second, partial matrix.
in	<i>b_end_row_ind</i>	row end of the second, partial matrix.
in	<i>b_start_col_ind</i>	column begin of the second, partial matrix.
in	<i>b_end_col_ind</i>	column end of the second, partial matrix.
in	<i>dest_col_size</i>	column size of the destination matrix.
out	<i>dest_matrix</i> [][]	pointer to the destination matrix.

Definition at line 391 of file matrix.c.

Referenced by `matrix_test()`.

#### 10.42.3.32 `matrix_part_mul_scalar_vec_matr()`

```
void matrix_part_mul_scalar_vec_matr (
    uint8_t max_m,
    uint8_t max_n,
    matrix_t scalar,
    matrix_t vec[max_m],
    matrix_t matrix[max_m][max_n],
    uint8_t begin_row,
    uint8_t begin_column,
    matrix_t dst_arr[max_n - begin_row] )
```

Compute the multiplication of a scalar with row vector and a sub-matrix.

Return  $\text{scal} * R_{1,m} * A_{m,n} = R_{1,n}$ , where *scal* is a scalar, *R* is a *m*-dimensional row vector, *A* is a (*m*×*n*)-matrix, and the result is a row vector of *n*-dimension.

## Parameters

in	<i>max_m</i>	size of the row vector and row number of the matrix.
in	<i>max_n</i>	column number of the matrix.
in	<i>scalar</i>	scalar value.
in	<i>vec</i> []	pointer to the row vector.
in	<i>matrix</i> [][]	pointer to the matrix.
in	<i>begin_row</i>	start row number of the sub-matrix.
in	<i>begin_column</i>	start column number of the sub-matrix.
out	<i>dst_arr</i>	pointer to the destination row vector of <i>n</i> -dimension.

Definition at line 489 of file matrix.c.

References `matrix_t`.

### 10.42.3.33 `matrix_part_print()`

```
void matrix_part_print (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t start_row_ind,
    uint8_t end_row_ind,
    uint8_t start_col_ind,
    uint8_t end_col_ind )
```

Display the values of the sub-matrix elements.

#### Parameters

in	<i>m</i>	total row number of the matrix.
in	<i>n</i>	total column number of the matrix.
in	<i>matrix</i> [][]	pointer to the entire matrix.
in	<i>start_row_ind</i>	start row number of the sub-matrix.
in	<i>end_row_ind</i>	end row number of the sub-matrix.
in	<i>start_col_ind</i>	start column number of the sub-matrix.
in	<i>end_col_ind</i>	end column number of the sub-matrix.

Definition at line 164 of file matrix.c.

### 10.42.3.34 `matrix_part_swap_rows()`

```
void matrix_part_swap_rows (
    uint8_t n,
    matrix_t matrix[][n],
    uint8_t i,
    uint8_t j,
    uint8_t col_begin,
    uint8_t col_end )
```

Swaps two rows of a sub-matrix.

Swaps the rows *i* and *j* of a part of a matrix.

**Parameters**

in	<i>n</i>	column number of the entire matrix.
in, out	<i>matrix</i> [][]	pointer to the entire matrix.
in	<i>i</i>	the i-the row of the sub-matrix.
in	<i>j</i>	the j-the row of the sub-matrix.
in	<i>col_begin</i>	the column begin of the sub-matrix.
in	<i>col_end</i>	the column end of the sub-matrix.

Definition at line 736 of file matrix.c.

References `matrix_t`.

Referenced by `lu_decomp()`.

**10.42.3.35 matrix\_print()**

```
void matrix_print (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n] )
```

Display the values of the matrix elements.

**Parameters**

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix</i> [][]	pointer to the entire matrix.

Definition at line 141 of file matrix.c.

Referenced by `matrix_get_two_norm()`, and `svd_compute_print_U_S_V_s()`.

**10.42.3.36 matrix\_read()**

```
matrix_t matrix_read (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t i,
    uint8_t j )
```

Get the value of a matrix at the position (i,j).

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>matrix</i> [][]	pointer to the matrix.
in	<i>i</i>	row number.
in	<i>j</i>	column number.

## Returns

the value of the matrix at the row *i* and column *j*.

Definition at line 895 of file matrix.c.

## 10.42.3.37 matrix\_set\_diag\_elements()

```
void matrix_set_diag_elements (
    uint8_t m,
    uint8_t n,
    matrix_t value,
    matrix_t diag_matrix[m][n] )
```

Set all the diagonal elements of a matrix with a specified value.

## Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>value</i>	value to be set.
in, out	<i>diag_matrix</i> [][]	pointer to the matrix.

Definition at line 565 of file matrix.c.

Referenced by matrix\_get\_diag\_mat().

## 10.42.3.38 matrix\_sub()

```
void matrix_sub (
    uint8_t m,
    uint8_t n,
    matrix_t A[m][n],
    matrix_t B[m][n],
    matrix_t A_minus_B[m][n] )
```

Compute the subtraction of two matrices.

Subtract matrix B from matrix A and return the result in A\_minus\_B matrix.

## Parameters

in	$m$	row number of the matrix to add.
in	$n$	column number of the matrix to add.
in	$A[[]]$	pointer to the first matrix.
in	$B[[]]$	pointer to the second matrix.
out	$A\_minus\_B[[]]$	pointer to the destination matrix.

Definition at line 329 of file matrix.c.

Referenced by matrix\_test().

### 10.42.3.39 matrix\_swap\_rows()

```
void matrix_swap_rows (
    uint8_t n,
    matrix_t matrix[] [n],
    uint8_t i,
    uint8_t j )
```

Swaps two rows of a matrix.

Swaps the rows  $i$  and  $j$  of a matrix.

## Parameters

in	$n$	column number of the matrix.
in	$matrix[[]]$	pointer to the matrix.
in	$i$	the $i$ -the row of the matrix.
in	$j$	the $j$ -the row of the matrix.

Definition at line 725 of file matrix.c.

References matrix\_t.

Referenced by lu\_decomp(), and matrix\_test().

### 10.42.3.40 matrix\_trans\_mul\_itself()

```
void matrix_trans_mul_itself (
    uint8_t m,
    uint8_t n,
    matrix_t A[m] [n],
    matrix_t AT_mul_A[n] [n] )
```

Compute the multiplication of the transpose of a matrix with itself.

Transpose the matrix  $A$  and multiply it with the matrix  $A$ :  $A' * A$ .

## Parameters

in	$m$	row number of the matrix.
in	$n$	column number of the matrix.
in	$A[[]]$	pointer to the matrix.
out	$AT\_mul\_A[[]]$	pointer to the destination matrix ( $A^t * A$ ).

Definition at line 545 of file matrix.c.

References `matrix_get_column_vec()`, and `matrix_t`.

Referenced by `magnetic_based_jacobian_get_JTJ()`, `modified_gauss_newton()`, `opt_levenberg_marquardt()`, and `opt_levenberg_marquardt_correction()`.

**10.42.3.41 matrix\_trans\_mul\_vec()**

```
void matrix_trans_mul_vec (
    uint8_t m,
    uint8_t n,
    matrix_t A[m][n],
    uint8_t b_size,
    matrix_t b_vec[m],
    matrix_t c_vec[n] )
```

Compute the multiplication of transposed matrix with column vector.

Transpose A and return  $A^t n, m * V_{m,1} = B_{n,1}$ , where A is a (mxn)-matrix, V is a m-dimensional column vector, and the result is a n-dimensional column vector.

## Parameters

in	$m$	row number of the matrix to transpose and multiply.
in	$n$	column number of the matrix to transpose and multiply.
in	$A[[]]$	pointer to the matrix.
in	$b\_size$	size of the column vector.
in	$b\_vec[[]]$	pointer to the column vector of m-dimension.
out	$c\_vec[[]]$	pointer to the destination, column vector of n-dimension.

Definition at line 511 of file matrix.c.

Referenced by `magnetic_based_jacobian_get_JTf()`, `modified_gauss_newton()`, `opt_levenberg_marquardt()`, `opt_levenberg_marquardt_correction()`, `solve_givens()`, and `solve_householder()`.

**10.42.3.42 matrix\_transpose()**

```
void matrix_transpose (
    uint8_t m,
```

```

uint8_t n,
matrix_t src_matrix[m][n],
matrix_t dest_matrix[n][m] )

```

Computes the transpose of a matrix.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>src_matrix</i> [][]	pointer to the matrix to transpose.
out	<i>dest_matrix</i> [][]	pointer to the destination matrix.

Definition at line 56 of file matrix.c.

Referenced by `get_PDOP()`, `matrix_test()`, `moore_penrose_get_pinv()`, and `triangular_matrices_test()`.

#### 10.42.3.43 matrix\_vec\_mul\_matr()

```

void matrix_vec_mul_matr (
    uint8_t m,
    uint8_t n,
    matrix_t vec[m],
    matrix_t matrix[m][n],
    matrix_t dst_arr[n] )

```

Compute the multiplication of row vector and a matrix.

Return  $R_{1,m} * A_{m,n} = R_{1,n}$ , where  $R$  is a  $m$ -dimensional row vector,  $A$  is a  $(m \times n)$ -matrix, and the result is a row vector of  $n$ -dimension.

#### Parameters

in	<i>m</i>	size of the row vector and row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>vec</i> []	pointer to the row vector.
in	<i>matrix</i> [][]	pointer to the matrix.
out	<i>dst_arr</i>	pointer to the destination row vector of $n$ -dimension.

Definition at line 452 of file matrix.c.

References `matrix_t`.

#### 10.42.3.44 matrix\_write()

```

void matrix_write (
    uint8_t m,

```

```

uint8_t n,
matrix_t matrix[m][n],
uint8_t i,
uint8_t j,
matrix_t val )

```

Write a value in a matrix at the position (i,j).

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
out	<i>matrix[ ][ ]</i>	pointer to the matrix.
in	<i>i</i>	row number.
in	<i>j</i>	column number.
in	<i>val</i>	value to write in the matrix.

Definition at line 901 of file matrix.c.

## 10.43 matrix\_test.c File Reference

Examples of matrix computations.

```

#include <stdio.h>
#include <stdint.h>
#include <inttypes.h>
#include <math.h>
#include "utils.h"
#include "matrix.h"

```

### Functions

- void [matrix\\_test](#) (void)  
*Test some matrix operations.*
- void [inv\\_triangular\\_matrices\\_test](#) (void)  
*Examples of the inverse of triangular matrices.*
- void [triangular\\_matrices\\_test](#) (void)  
*Examples of triangular matrices.*

#### 10.43.1 Detailed Description

Examples of matrix computations.

Matrix computation examples of the (see [matrix](#) functions).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.44 matrix\_test.h File Reference

Examples of matrix computations.

### Functions

- void [matrix\\_test](#) (void)  
*Test some matrix operations.*
- void [inv\\_triangular\\_matrices\\_test](#) (void)  
*Examples of the inverse of triangular matrices.*
- void [triangular\\_matrices\\_test](#) (void)  
*Examples of triangular matrices.*

### 10.44.1 Detailed Description

Examples of matrix computations.

Matrix computation examples (see [matrix](#) functions).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.45 modified\_gauss\_newton.c File Reference

Implement the Gauss–Newton algorithm.

```
#include <stdio.h>
#include <math.h>
#include "utils.h"
#include "matrix.h"
#include "vector.h"
#include "moore_penrose_pseudo_inverse.h"
```

### Functions

- uint8\_t [modified\\_gauss\\_newton](#) (uint8\_t f\_length, uint8\_t n, [vector\\_t](#) x0\_vec[n], [vector\\_t](#) data\_vec[f\_length], [matrix\\_t](#) eps, [matrix\\_t](#) fmin, uint8\_t max\_iter\_num, [vector\\_t](#) est\_x\_vec[n], void(\*get\_f\_error)([vector\\_t](#) x0\_vec[], [vector\\_t](#) data\_vec[], [vector\\_t](#) f\_vec[]), void(\*get\_jacobian)([vector\\_t](#) x0\_vec[], [matrix\\_t](#) J[][n]))  
*Implements the modified Gauss–Newton algorithm.*

## 10.45.1 Detailed Description

Implement the Gauss–Newton algorithm.

### Note

This function is generally implemented.

### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

## 10.45.2 Function Documentation

### 10.45.2.1 modified\_gauss\_newton()

```
uint8_t modified_gauss_newton (
    uint8_t f_length,
    uint8_t n,
    vector_t x0_vec[n],
    vector_t data_vec[f_length],
    matrix_t eps,
    matrix_t fmin,
    uint8_t max_iter_num,
    vector_t est_x_vec[n],
    void(*) (vector_t x0_vec[], vector_t data_vec[], vector_t f_vec[]) get_f_error,
    void(*) (vector_t x0_vec[], matrix_t J[][n]) get_jacobian )
```

Implements the modified Gauss–Newton algorithm.

The user should provide pointers to the error and Jacobian functions.

### Note

This function is generally implemented.

## Parameters

in	<i>f_length</i>	length of the error functions vector.
in	<i>n</i>	length of the start vector.
in	<i>x0_vec[]</i>	start vector.
in	<i>data_vec[]</i>	data vector.
in	<i>eps</i>	accuracy bound.
in	<i>fmin</i>	termination tolerance on the error function.
in	<i>max_iter_num</i>	maximal iteration number of the Gauss–Newton algorithm.
out	<i>est_x_vec[]</i>	estimated (optimized) vector.
in	<i>(*get_f_error)</i>	pointer to the error function.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

## Returns

required iteration number.

Definition at line 32 of file `modified_gauss_newton.c`.

References `matrix_mul_vec()`, `matrix_t`, `matrix_trans_mul_itself()`, `matrix_trans_mul_vec()`, `moore_penrose_↔get_pinv()`, `utils_max()`, `vector_copy()`, `vector_get_euclidean_distance()`, `vector_get_norm2()`, `vector_sub()`, and `vector_t`.

Referenced by `optimization_exponential_data_test()`, `optimization_sinusoidal_data_test()`, and `optimization_test()`.

## 10.46 modified\_gauss\_newton.h File Reference

Implement the Gauss–Newton algorithm.

```
#include <inttypes.h>
#include "matrix.h"
#include "vector.h"
```

### Functions

- `uint8_t modified_gauss_newton` (`uint8_t f_length`, `uint8_t n`, `vector_t x0_vec[n]`, `vector_t data_vec[f_length]`, `matrix_t eps`, `matrix_t fmin`, `uint8_t max_iter_num`, `vector_t est_x_vec[n]`, `void(*get_f_error)(vector_t x0_↔vec[], vector_t data_vec[], vector_t f_vec[])`, `void(*get_jacobian)(vector_t x0_vec[], matrix_t J[][n])`)

*Implements the modified Gauss–Newton algorithm.*

### 10.46.1 Detailed Description

Implement the Gauss–Newton algorithm.

#### Note

This function is generally implemented.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
 Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

## 10.46.2 Function Documentation

### 10.46.2.1 modified\_gauss\_newton()

```
uint8_t modified_gauss_newton (
    uint8_t f_length,
    uint8_t n,
    vector_t x0_vec[n],
    vector_t data_vec[f_length],
    matrix_t eps,
    matrix_t fmin,
    uint8_t max_iter_num,
    vector_t est_x_vec[n],
    void(*) (vector_t x0_vec[], vector_t data_vec[], vector_t f_vec[]) get_f_error,
    void(*) (vector_t x0_vec[], matrix_t J[][n]) get_jacobian )
```

Implements the modified Gauss–Newton algorithm.

The user should provide pointers to the error and Jacobian functions.

#### Note

This function is generally implemented.

#### Parameters

in	<i>f_length</i>	length of the error functions vector.
in	<i>n</i>	length of the start vector.
in	<i>x0_vec[]</i>	start vector.
in	<i>data_vec[]</i>	data vector.
in	<i>eps</i>	accuracy bound.
in	<i>fmin</i>	termination tolerance on the error function.
in	<i>max_iter_num</i>	maximal iteration number of the Gauss–Newton algorithm.
out	<i>est_x_vec[]</i>	estimated (optimized) vector.
in	<i>(*get_f_error)</i>	pointer to the error function.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

#### Returns

required iteration number.

Definition at line 32 of file modified\_gauss\_newton.c.

References `matrix_mul_vec()`, `matrix_t`, `matrix_trans_mul_itself()`, `matrix_trans_mul_vec()`, `moore_penrose`  $\leftrightarrow$  `get_pinv()`, `utils_max()`, `vector_copy()`, `vector_get_euclidean_distance()`, `vector_get_norm2()`, `vector_sub()`, and `vector_t`.

Referenced by `optimization_exponential_data_test()`, `optimization_sinusoidal_data_test()`, and `optimization_test()`.

## 10.47 moore\_penrose\_pinv\_test.c File Reference

Examples of the Moore–Penrose algorithm.

```
#include <stdio.h>
#include <inttypes.h>
#include "moore_penrose_pseudo_inverse.h"
#include "matrix.h"
```

### Functions

- void [moore\\_penrose\\_pinv\\_test](#) (void)  
*Examples of the Moore–Penrose algorithm.*

#### 10.47.1 Detailed Description

Examples of the Moore–Penrose algorithm.

Moore–Penrose algorithm examples (see the [Moore–Penrose](#) approach).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.48 moore\_penrose\_pinv\_test.h File Reference

Examples of the Moore–Penrose algorithm.

### Functions

- void [moore\\_penrose\\_pinv\\_test](#) (void)  
*Examples of the Moore–Penrose algorithm.*

#### 10.48.1 Detailed Description

Examples of the Moore–Penrose algorithm.

Moore–Penrose algorithm examples (see the [Moore–Penrose](#) approach).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.49 moore\_penrose\_pseudo\_inverse.c File Reference

Moore–Penrose algorithm to compute the pseudo-inverse of a rectangular matrix.

```
#include <stdbool.h>
#include <stdio.h>
#include "moore_penrose_pseudo_inverse.h"
#include "matrix.h"
#include "svd.h"
```

### Functions

- `int8_t moore_penrose_get_pinv` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], `matrix_t` pinv\_A[n][m])  
*Calculate the Moore–Penrose inverse of a rectangular matrix.*
- `void moore_penrose_pinv_compute_print` (uint8\_t m, uint8\_t n, `matrix_t` matrix[m][n], uint8\_t i)  
*Compute and print the Moore–Penrose inverse of a matrix.*

### 10.49.1 Detailed Description

Moore–Penrose algorithm to compute the pseudo-inverse of a rectangular matrix.

The computation of the pseudo-inverse is based on the Singular Value Decomposition (SVD).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.49.2 Function Documentation

#### 10.49.2.1 moore\_penrose\_get\_pinv()

```
int8_t moore_penrose_get_pinv (
    uint8_t m,
    uint8_t n,
    matrix_t A[m][n],
    matrix_t pinv_A[n][m] )
```

Calculate the Moore–Penrose inverse of a rectangular matrix.

The computation of the Moore–Penrose inverse is based on the Golub–Kahan–Reinsch SVD algorithm.

**Parameters**

in	$m$	row number of the matrix to inverse.
in	$n$	column number of the matrix to inverse.
in	$A[[]]$	pointer to the matrix A.
out	$pinv\_A[[]]$	pointer to the pseudo-inverse matrix.

**Returns**

- 1, if the computation of the Moore-Penrose inverse is successful.
- 1, if the maximal, allowed column or row number is exceeded.
- 2, if the matrix is underdetermined ( $m < n$ ).
- 3, if the rank of the matrix is equal to 0.

Definition at line 38 of file moore\_penrose\_pseudo\_inverse.c.

References `matrix_dim_t::col_num`, `matrix_t`, `matrix_transpose()`, `matrix_dim_t::row_num`, `svd_get_single_↵`  
`values_num()`, and `svd_get_U_dim()`.

Referenced by `get_delta_x()`, `get_PDOP()`, `loc_gauss_newton()`, `modified_gauss_newton()`, `moore_penrose_↵`  
`pinv_compute_print()`, `newton_raphson()`, `solve()`, `trilateration2()`, and `trilateration_get_particular_solution()`.

**10.49.2.2 moore\_penrose\_pinv\_compute\_print()**

```
void moore_penrose_pinv_compute_print (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t i )
```

Compute and print the Moore–Penrose inverse of a matrix.

**Parameters**

in	$m$	row number of the matrix.
in	$n$	column number of the matrix.
in	$matrix[[]]$	pointer to the matrix.
in	$i$	label.

Definition at line 167 of file moore\_penrose\_pseudo\_inverse.c.

References `matrix_flex_print()`, `matrix_t`, and `moore_penrose_get_pinv()`.

Referenced by `moore_penrose_pinv_test()`.

**10.50 moore\_penrose\_pseudo\_inverse.h File Reference**

Moore–Penrose algorithm to compute the pseudo-inverse of a matrix.

```
#include "matrix.h"
```

## Macros

- `#define MAX_ROW_NUM 23`  
*The maximal row number allowed.*
- `#define MAX_COL_NUM 23`  
*The maximal column number allowed.*
- `#define MOORE_PENROSE_PSEUDO_COMP_SUCCESS 1`  
*The Moore–Penrose inverse is successfully completed.*
- `#define MOORE_PENROSE_PSEUDO_MAX_ALLOW_ROW_COL_EXCEED -1`  
*The maximal row number allowed is exceeded.*
- `#define MOORE_PENROSE_PSEUDO_GIVE_MATRIX_TRANSPOSE -2`  
*The transposed matrix should be delivered.*
- `#define MOORE_PENROSE_INVALID_RANK_VALUE -3`  
*Invalid rank value of a matrix.*

## Functions

- `int8_t moore_penrose_get_pinv (uint8_t m, uint8_t n, matrix_t A[m][n], matrix_t pinv_A[n][m])`  
*Calculate the Moore–Penrose inverse of a rectangular matrix.*
- `void moore_penrose_pinv_compute_print (uint8_t m, uint8_t n, matrix_t matrix[m][n], uint8_t i)`  
*Compute and print the Moore–Penrose inverse of a matrix.*

### 10.50.1 Detailed Description

Moore–Penrose algorithm to compute the pseudo-inverse of a matrix.

The computation of the pseudo-inverse is based on the Singular Value Decomposition (SVD).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.50.2 Function Documentation

#### 10.50.2.1 moore\_penrose\_get\_pinv()

```
int8_t moore_penrose_get_pinv (
    uint8_t m,
    uint8_t n,
    matrix_t A[m][n],
    matrix_t pinv_A[n][m] )
```

Calculate the Moore–Penrose inverse of a rectangular matrix.

The computation of the Moore–Penrose inverse is based on the Golub–Kahan–Reinsch SVD algorithm.

**Parameters**

in	$m$	row number of the matrix to inverse.
in	$n$	column number of the matrix to inverse.
in	$A[[]]$	pointer to the matrix A.
out	$pinv\_A[[]]$	pointer to the pseudo-inverse matrix.

**Returns**

- 1, if the computation of the Moore-Penrose inverse is successful.
- 1, if the maximal, allowed column or row number is exceeded.
- 2, if the matrix is underdetermined ( $m < n$ ).
- 3, if the rank of the matrix is equal to 0.

Definition at line 38 of file moore\_penrose\_pseudo\_inverse.c.

References `matrix_dim_t::col_num`, `matrix_t`, `matrix_transpose()`, `matrix_dim_t::row_num`, `svd_get_single_↵`  
`values_num()`, and `svd_get_U_dim()`.

Referenced by `get_delta_x()`, `get_PDOP()`, `loc_gauss_newton()`, `modified_gauss_newton()`, `moore_penrose_↵`  
`pinv_compute_print()`, `newton_raphson()`, `solve()`, `trilateration2()`, and `trilateration_get_particular_solution()`.

**10.50.2.2 moore\_penrose\_pinv\_compute\_print()**

```
void moore_penrose_pinv_compute_print (
    uint8_t m,
    uint8_t n,
    matrix_t matrix[m][n],
    uint8_t i )
```

Compute and print the Moore–Penrose inverse of a matrix.

**Parameters**

in	$m$	row number of the matrix.
in	$n$	column number of the matrix.
in	$matrix[[]]$	pointer to the matrix.
in	$i$	label.

Definition at line 167 of file moore\_penrose\_pseudo\_inverse.c.

References `matrix_flex_print()`, `matrix_t`, and `moore_penrose_get_pinv()`.

Referenced by `moore_penrose_pinv_test()`.

**10.51 multipath\_algo\_own\_norm\_distr\_test.c File Reference**

Example of the algorithm for the recognition and mitigation of multipath effects.

```
#include <float.h>
#include <stdio.h>
#include <unistd.h>
#include <inttypes.h>
#include "multipath_dist_detection_mitigation.h"
#include "loc_levenberg_marquardt.h"
#include "norm_dist_rnd_generator.h"
#include "dist_based_jacobian.h"
#include "dist_based_fi.h"
#include "shell_sort.h"
#include "vector.h"
#include "DOP.h"
```

## Functions

- void [multipath\\_algo\\_own\\_norm\\_distr\\_test](#) (void)  
*Example of the algorithm for the recognition and mitigation of multipath effects.*

### 10.51.1 Detailed Description

Example of the algorithm for the recognition and mitigation of multipath effects.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
Naouar Guerchali  
Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

## 10.52 multipath\_algo\_own\_norm\_distr\_test.h File Reference

Example of the algorithm for the recognition and mitigation of multipath effects.

## Functions

- void [multipath\\_algo\\_own\\_norm\\_distr\\_test](#) (void)  
*Example of the algorithm for the recognition and mitigation of multipath effects.*

### 10.52.1 Detailed Description

Example of the algorithm for the recognition and mitigation of multipath effects.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
Naouar Guerchali  
Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

## 10.53 multipath\_dist\_detection\_mitigation.c File Reference

Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm.

```
#include <math.h>
#include <stdbool.h>
#include <stdio.h>
#include <float.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include "matrix.h"
#include "vector.h"
#include "combinatorics.h"
#include "multipath_dist_detection_mitigation.h"
#include "norm_dist_rnd_generator.h"
#include "shell_sort.h"
#include "dist_based_position.h"
#include "DOP.h"
#include "trilateration.h"
```

### Functions

- [matrix\\_t get\\_exact\\_distance\\_to\\_anchor](#) ([matrix\\_t](#) ref\_point[], [uint32\\_t](#) exact\_point[])  
*Computes the exact distance between a mobile station and a reference station.*
- [bool is\\_member](#) ([matrix\\_t](#) vector, [uint8\\_t](#) n, [matrix\\_t](#) multipath[n])  
*Determine if a candidate is a multipath or not.*
- [bool is\\_anchor](#) ([uint8\\_t](#) m, [matrix\\_t](#) ref\_matr[m][3], [uint32\\_t](#) point[3])  
*Determine if a point is an anchor or not.*
- [void sim\\_UWB\\_dist](#) ([uint8\\_t](#) m, [matrix\\_t](#) ref\_matrix[m][3], [uint32\\_t](#) exact\_point[], [matrix\\_t](#) sigma, [uint8\\_t](#) n, [matrix\\_t](#) multipath[n], [int](#) seed, [matrix\\_t](#) r\_noised\_vec[])  
*Simulate an UWB-based localization system.*
- [void get\\_optimal\\_partial\\_ref\\_matrix](#) ([uint8\\_t](#) anchors\_num, [matrix\\_t](#) ref\_matrix[anchors\_num][3], [uint8\\_t](#) k, [uint8\\_t](#) optimal\_anchors\_comb[k], [matrix\\_t](#) opt\_partial\_ref\_matrix[k][3])  
*Compute the optimal partial matrix including reference points.*
- [void get\\_optimal\\_partial\\_r\\_noised\\_vec](#) ([uint8\\_t](#) k, [matrix\\_t](#) r\_noised\_vec[], [uint8\\_t](#) optimal\_anchors\_comb[k], [matrix\\_t](#) opt\_sub\_r\_noised\_vec[k])  
*Compute noised distances corresponding to the optimal partial matrix.*
- [void recog\\_mitigate\\_multipath](#) ([uint8\\_t](#) k, [uint8\\_t](#) m, [matrix\\_t](#) ref\_matrix[m][3], [matrix\\_t](#) noised\_r\_vec[m], [uint8\\_t](#) anchors\_optimal\_combi[k], [matrix\\_t](#) start\_optimal\_pos[3])  
*Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm.*

### 10.53.1 Detailed Description

Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm.

#### Author

Zakaria Kasmi   [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
 Abdelmoumen Norrdine   [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)  
 Naouar Guerchali

## 10.53.2 Function Documentation

### 10.53.2.1 get\_exact\_distance\_to\_anchor()

```
matrix_t get_exact_distance_to_anchor (
    matrix_t ref_point[],
    uint32_t exact_point[] )
```

Computes the exact distance between a mobile station and a reference station.

#### Parameters

in	<i>ref_point</i>	three-dimensional coordinates of a reference station.
in	<i>exact_point</i>	three-dimensional coordinates of the mobile device.

#### Returns

the exact distance between the mobile station and the reference station.

Definition at line 42 of file multipath\_dist\_detection\_mitigation.c.

References `matrix_t`.

Referenced by `sim_UWB_dist()`.

### 10.53.2.2 get\_optimal\_partial\_r\_noised\_vec()

```
void get_optimal_partial_r_noised_vec (
    uint8_t k,
    matrix_t r_noised_vec[],
    uint8_t optimal_anchors_comb[k],
    matrix_t opt_sub_r_noised_vec[k] )
```

Compute noised distances corresponding to the optimal partial matrix.

#### Parameters

in	<i>k</i>	k anchors (possible sub-experiments).
in	<i>r_noised_vec[]</i>	include noised distances.
in	<i>optimal_anchors_comb[]</i>	optimal k-anchor combination.
in, out	<i>opt_sub_r_noised_vec[][]</i>	noised distances corresponding to the optimal k-anchors.

Definition at line 123 of file multipath\_dist\_detection\_mitigation.c.

Referenced by `multipath_algo_own_norm_distr_test()`, and `recog_mitigate_multipath()`.

### 10.53.2.3 `get_optimal_partial_ref_matrix()`

```
void get_optimal_partial_ref_matrix (
    uint8_t anchors_num,
    matrix_t ref_matrix[anchors_num][3],
    uint8_t k,
    uint8_t optimal_anchors_comb[k],
    matrix_t opt_partial_ref_matrix[k][3] )
```

Compute the optimal partial matrix including reference points.

#### Parameters

in	<i>anchors_num</i>	number of the reference points.
in	<i>ref_matrix[[]]</i>	three-dimensional coordinates of the reference stations.
in	<i>k</i>	k anchors (possible sub-experiments).
in	<i>optimal_anchors_comb[[]]</i>	optimal anchor combination.
out	<i>opt_partial_ref_matrix[[]]</i>	optimal partial matrix of anchors.

Definition at line 108 of file `multipath_dist_detection_mitigation.c`.

Referenced by `multipath_algo_own_norm_distr_test()`, and `recog_mitigate_multipath()`.

### 10.53.2.4 `is_anchor()`

```
bool is_anchor (
    uint8_t m,
    matrix_t ref_matr[m][3],
    uint32_t point[3] )
```

Determine if a point is an anchor or not.

**Parameters**

in	<i>m</i>	number of the reference points.
in	<i>ref_matr</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of a point.

**Returns**

true, if a point is an anchor.  
false, if not.

Definition at line 65 of file multipath\_dist\_detection\_mitigation.c.

Referenced by multipath\_algo\_own\_norm\_distr\_test().

**10.53.2.5 is\_member()**

```
bool is_member (
    matrix_t vector,
    uint8_t n,
    matrix_t multipath[n] )
```

Determine if a candidate is a multipath or not.

**Parameters**

in	<i>vector</i>	a candidate.
in	<i>n</i>	size of the multipath-vector.
in	<i>multipath</i> []	pointer to the multipath-vector.

**Returns**

true, if a candidate is a multipath.  
false, if not.

Definition at line 55 of file multipath\_dist\_detection\_mitigation.c.

Referenced by sim\_UWB\_dist().

**10.53.2.6 recog\_mitigate\_multipath()**

```
void recog_mitigate_multipath (
    uint8_t k,
    uint8_t m,
    matrix_t ref_Matrix[m][3],
```

```
matrix_t r_noised_vec[m],  
uint8_t anchors_optimal[k],  
matrix_t start_optimal[3] )
```

Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm.

## Parameters

in	<i>k</i>	k anchors (possible sub-experiments).
in	<i>m</i>	number of anchors.
in	<i>ref_Matrix</i> [[[]]	three-dimensional coordinates of the reference stations.
in	<i>r_noised_vec</i> []	include noised distances.
in	<i>anchors_optimal</i> []	optimal k-anchor combination.
in, out	<i>start_optimal</i> []	optimal position.

Definition at line 133 of file multipath\_dist\_detection\_mitigation.c.

References COMBI\_ERROR, COMBI\_SUCCESS, combinatorics\_get\_next\_without\_rep(), combinatorics\_init(), get\_optimal\_partial\_r\_noised\_vec(), get\_optimal\_partial\_ref\_matrix(), matrix\_t, shell\_sort(), trilateration2(), vector\_copy(), vector\_get\_index\_vector(), vector\_get\_norm2(), vector\_square(), and vector\_sub().

Referenced by multipath\_algo\_own\_norm\_distr\_test().

## 10.53.2.7 sim\_UWB\_dist()

```
void sim_UWB_dist (
    uint8_t m,
    matrix_t ref_matrix[m][3],
    uint32_t exact_point[],
    matrix_t sigma,
    uint8_t n,
    matrix_t multipath[n],
    int seed,
    matrix_t r_noised_vec[] )
```

Simulate an UWB-based localization system.

The UWB system is simulated by noising the distances from a mobile to the reference station.

## Parameters

in	<i>m</i>	number of the reference points.
in	<i>ref_matrix</i> [[[]]	three-dimensional coordinates of the reference stations.
in	<i>exact_point</i> []	three-dimensional coordinates of a true position.
in	<i>sigma</i>	$\sigma_{M,W}$ parameter.
in	<i>n</i>	number of the multipath-affected distances.
in	<i>multipath</i> []	multipath vector.
in	<i>seed</i>	seed value.
in	<i>r_noised_vec</i> []	noised distances.

Definition at line 78 of file multipath\_dist\_detection\_mitigation.c.

References get\_exact\_distance\_to\_anchor(), get\_norm\_distr\_rand\_num(), get\_rand\_num(), is\_member(), and matrix\_t.

Referenced by multipath\_algo\_own\_norm\_distr\_test().

## 10.54 multipath\_dist\_detection\_mitigation.h File Reference

Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm.

```
#include <inttypes.h>
#include "matrix.h"
```

### Functions

- [matrix\\_t get\\_exact\\_distance\\_to\\_anchor](#) ([matrix\\_t](#) ref\_point[], [uint32\\_t](#) exact\_point[])  
*Computes the exact distance between a mobile station and a reference station.*
- [bool is\\_member](#) ([matrix\\_t](#) vector, [uint8\\_t](#) n, [matrix\\_t](#) multipath[n])  
*Determine if a candidate is a multipath or not.*
- [bool is\\_anchor](#) ([uint8\\_t](#) m, [matrix\\_t](#) ref\_matr[m][3], [uint32\\_t](#) point[3])  
*Determine if a point is an anchor or not.*
- [void sim\\_UWB\\_dist](#) ([uint8\\_t](#) m, [matrix\\_t](#) ref\_matrix[m][3], [uint32\\_t](#) exact\_point[], [matrix\\_t](#) sigma, [uint8\\_t](#) n, [matrix\\_t](#) multipath[n], [int](#) seed, [matrix\\_t](#) r\_noised\_vec[])  
*Simulate an UWB-based localization system.*
- [void get\\_optimal\\_partial\\_ref\\_matrix](#) ([uint8\\_t](#) anchors\_num, [matrix\\_t](#) ref\_matrix[anchors\_num][3], [uint8\\_t](#) k, [uint8\\_t](#) optimal\_anchors\_comb[k], [matrix\\_t](#) opt\_partial\_ref\_matrix[k][3])  
*Compute the optimal partial matrix including reference points.*
- [void get\\_optimal\\_partial\\_r\\_noised\\_vec](#) ([uint8\\_t](#) k, [matrix\\_t](#) r\_noised\_vec[], [uint8\\_t](#) optimal\_anchors\_comb[k], [matrix\\_t](#) opt\_sub\_r\_noised\_vec[k])  
*Compute noised distances corresponding to the optimal partial matrix.*
- [void recog\\_mitigate\\_multipath](#) ([uint8\\_t](#) k, [uint8\\_t](#) m, [matrix\\_t](#) ref\_Matrix[m][3], [matrix\\_t](#) r\_noised\_vec[m], [uint8\\_t](#) anchors\_optimal[k], [matrix\\_t](#) start\_optimal[3])  
*Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm.*

### 10.54.1 Detailed Description

Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
 Abdelmoumen Norrdine [a.norrdine@googlemail.com](mailto:a.norrdine@googlemail.com)  
 Naouar Guerchali

### 10.54.2 Function Documentation

#### 10.54.2.1 get\_exact\_distance\_to\_anchor()

```
matrix\_t get_exact_distance_to_anchor (
    matrix\_t ref_point[],
    uint32\_t exact_point[] )
```

Computes the exact distance between a mobile station and a reference station.

## Parameters

in	<i>ref_point</i>	three-dimensional coordinates of a reference station.
in	<i>exact_point</i>	three-dimensional coordinates of the mobile device.

## Returns

the exact distance between the mobile station and the reference station.

Definition at line 42 of file multipath\_dist\_detection\_mitigation.c.

References `matrix_t`.

Referenced by `sim_UWB_dist()`.

10.54.2.2 `get_optimal_partial_r_noised_vec()`

```
void get_optimal_partial_r_noised_vec (
    uint8_t k,
    matrix_t r_noised_vec[],
    uint8_t optimal_anchors_comb[k],
    matrix_t opt_sub_r_noised_vec[k] )
```

Compute noised distances corresponding to the optimal partial matrix.

## Parameters

in	<i>k</i>	k anchors (possible sub-experiments).
in	<i>r_noised_vec[]</i>	include noised distances.
in	<i>optimal_anchors_comb[]</i>	optimal k-anchor combination.
in, out	<i>opt_sub_r_noised_vec[][]</i>	noised distances corresponding to the optimal k-anchors.

Definition at line 123 of file multipath\_dist\_detection\_mitigation.c.

Referenced by `multipath_algo_own_norm_distr_test()`, and `recog_mitigate_multipath()`.

10.54.2.3 `get_optimal_partial_ref_matrix()`

```
void get_optimal_partial_ref_matrix (
    uint8_t anchors_num,
    matrix_t ref_matrix[anchors_num][3],
    uint8_t k,
    uint8_t optimal_anchors_comb[k],
    matrix_t opt_partial_ref_matrix[k][3] )
```

Compute the optimal partial matrix including reference points.

**Parameters**

in	<i>anchors_num</i>	number of the reference points.
in	<i>ref_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>k</i>	k anchors (possible sub-experiments).
in	<i>optimal_anchors_comb</i> []	optimal anchor combination.
out	<i>opt_partial_ref_matrix</i> [][]	optimal partial matrix of anchors.

Definition at line 108 of file `multipath_dist_detection_mitigation.c`.

Referenced by `multipath_algo_own_norm_distr_test()`, and `recog_mitigate_multipath()`.

**10.54.2.4 is\_anchor()**

```
bool is_anchor (
    uint8_t m,
    matrix_t ref_matr[m][3],
    uint32_t point[3] )
```

Determine if a point is an anchor or not.

**Parameters**

in	<i>m</i>	number of the reference points.
in	<i>ref_matr</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>point</i> []	three-dimensional coordinates of a point.

**Returns**

true, if a point is an anchor.

false, if not.

Definition at line 65 of file multipath\_dist\_detection\_mitigation.c.

Referenced by multipath\_algo\_own\_norm\_distr\_test().

**10.54.2.5 is\_member()**

```
bool is_member (
    matrix_t vector,
    uint8_t n,
    matrix_t multipath[n] )
```

Determine if a candidate is a multipath or not.

**Parameters**

in	<i>vector</i>	a candidate.
in	<i>n</i>	size of the multipath-vector.
in	<i>multipath</i> []	pointer to the multipath-vector.

**Returns**

true, if a candidate is a multipath.

false, if not.

Definition at line 55 of file multipath\_dist\_detection\_mitigation.c.

Referenced by sim\_UWB\_dist().

**10.54.2.6 recog\_mitigate\_multipath()**

```
void recog_mitigate_multipath (
    uint8_t k,
    uint8_t m,
    matrix_t ref_Matrix[m][3],
```

```
matrix_t r_noised_vec[m],  
uint8_t anchors_optimal[k],  
matrix_t start_optimal[3] )
```

Implement the Multipath Distance Detection and Mitigation (MDDM) algorithm.

## Parameters

in	<i>k</i>	k anchors (possible sub-experiments).
in	<i>m</i>	number of anchors.
in	<i>ref_Matrix</i> [[[]]	three-dimensional coordinates of the reference stations.
in	<i>r_noised_vec</i> []	include noised distances.
in	<i>anchors_optimal</i> []	optimal k-anchor combination.
in, out	<i>start_optimal</i> []	optimal position.

Definition at line 133 of file multipath\_dist\_detection\_mitigation.c.

References COMBI\_ERROR, COMBI\_SUCCESS, combinatorics\_get\_next\_without\_rep(), combinatorics\_init(), get\_optimal\_partial\_r\_noised\_vec(), get\_optimal\_partial\_ref\_matrix(), matrix\_t, shell\_sort(), trilateration2(), vector\_copy(), vector\_get\_index\_vector(), vector\_get\_norm2(), vector\_square(), and vector\_sub().

Referenced by multipath\_algo\_own\_norm\_distr\_test().

## 10.54.2.7 sim\_UWB\_dist()

```
void sim_UWB_dist (
    uint8_t m,
    matrix_t ref_matrix[m][3],
    uint32_t exact_point[],
    matrix_t sigma,
    uint8_t n,
    matrix_t multipath[n],
    int seed,
    matrix_t r_noised_vec[] )
```

Simulate an UWB-based localization system.

The UWB system is simulated by noising the distances from a mobile to the reference station.

## Parameters

in	<i>m</i>	number of the reference points.
in	<i>ref_matrix</i> [[[]]	three-dimensional coordinates of the reference stations.
in	<i>exact_point</i> []	three-dimensional coordinates of a true position.
in	<i>sigma</i>	$\sigma_{M,W}$ parameter.
in	<i>n</i>	number of the multipath-affected distances.
in	<i>multipath</i> []	multipath vector.
in	<i>seed</i>	seed value.
in	<i>r_noised_vec</i> []	noised distances.

Definition at line 78 of file multipath\_dist\_detection\_mitigation.c.

References get\_exact\_distance\_to\_anchor(), get\_norm\_distr\_rand\_num(), get\_rand\_num(), is\_member(), and matrix\_t.

Referenced by multipath\_algo\_own\_norm\_distr\_test().

## 10.55 newton\_raphson.c File Reference

Implement the Newton–Raphson algorithm.

```
#include "vector.h"
#include "matrix.h"
#include "moore_penrose_pseudo_inverse.h"
```

### Functions

- `uint8_t newton_raphson (uint8_t f_length, uint8_t n, vector\_t x0_arr[], double eps, uint8_t max_it_num, vector\_t est_x_arr[], void(*get_non_lin_sys)(vector\_t x_arr[], vector\_t f_vec[]), void(*get_jacobian)(vector\_t x_arr[], matrix\_t J[][n]))`

*Implements the Newton–Raphson algorithm.*

### 10.55.1 Detailed Description

Implement the Newton–Raphson algorithm.

The Newton–Raphson algorithm enables to solve multi-variant nonlinear equation systems.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.55.2 Function Documentation

#### 10.55.2.1 newton\_raphson()

```
uint8_t newton_raphson (
    uint8_t f_length,
    uint8_t n,
    vector\_t x0_arr[],
    double eps,
    uint8_t max_it_num,
    vector\_t est_x_arr[],
    void(*) (vector\_t x_arr[], vector\_t f_vec[]) get_non_lin_sys,
    void(*) (vector\_t x_arr[], matrix\_t J[][n]) get_jacobian )
```

Implements the Newton–Raphson algorithm.

The user should provide pointers to non-linear equation systems and Jacobian functions.

Note

This function is generally implemented.

## Parameters

in	<i>f_length</i>	length of the error functions vector.
in	<i>n</i>	length of the start vector.
in	<i>x0_arr[]</i>	start vector.
in	<i>eps</i>	accuracy bound.
in	<i>max_it_num</i>	maximal iteration number of the Newton–Raphson algorithm.
out	<i>est_x_arr[]</i>	estimated (solution) vector.
in	<i>(*get_non_lin_sys)</i>	pointer to non-linear equation systems.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

## Returns

required iteration number.

Definition at line 28 of file newton\_raphson.c.

References `matrix_mul_vec()`, `matrix_t`, `moore_penrose_get_pinv()`, `vector_copy()`, `vector_get_euclidean_distance()`, `vector_sub()`, and `vector_t`.

Referenced by `fsolve()`.

## 10.56 newton\_raphson.h File Reference

Implement the Newton–Raphson algorithm.

```
#include <inttypes.h>
#include "vector.h"
#include "matrix.h"
```

### Functions

- `uint8_t newton_raphson (uint8_t f_length, uint8_t n, vector_t x0_arr[], double eps, uint8_t max_it_num, vector_t est_x_arr[], void(*get_non_lin_sys)(vector_t x_arr[], vector_t f_vec[]), void(*get_jacobian)(vector_t x_arr[], matrix_t J[][n]))`

*Implements the Newton–Raphson algorithm.*

### 10.56.1 Detailed Description

Implement the Newton–Raphson algorithm.

The Newton–Raphson algorithm enables to solve multi-variant nonlinear equation systems.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.56.2 Function Documentation

### 10.56.2.1 newton\_raphson()

```
uint8_t newton_raphson (
    uint8_t f_length,
    uint8_t n,
    vector_t x0_arr[],
    double eps,
    uint8_t max_it_num,
    vector_t est_x_arr[],
    void(*) (vector_t x_arr[], vector_t f_vec[]) get_non_lin_sys,
    void(*) (vector_t x_arr[], matrix_t J[][n]) get_jacobian )
```

Implements the Newton–Raphson algorithm.

The user should provide pointers to non-linear equation systems and Jacobian functions.

#### Note

This function is generally implemented.

#### Parameters

in	<i>f_length</i>	length of the error functions vector.
in	<i>n</i>	length of the start vector.
in	<i>x0_arr[]</i>	start vector.
in	<i>eps</i>	accuracy bound.
in	<i>max_it_num</i>	maximal iteration number of the Newton–Raphson algorithm.
out	<i>est_x_arr[]</i>	estimated (solution) vector.
in	<i>(*get_non_lin_sys)</i>	pointer to non-linear equation systems.
in	<i>(*get_jacobian)</i>	pointer to the Jacobian matrix.

#### Returns

required iteration number.

Definition at line 28 of file newton\_raphson.c.

References `matrix_mul_vec()`, `matrix_t`, `moore_penrose_get_pinv()`, `vector_copy()`, `vector_get_euclidean_↵ distance()`, `vector_sub()`, and `vector_t`.

Referenced by `fsolve()`.

## 10.57 norm\_dist\_rnd\_generator.c File Reference

Generating normally distributed random numbers.

```
#include <stdlib.h>
#include <math.h>
#include "norm_dist_rnd_generator.h"
```

## Functions

- double [get\\_norm\\_distr\\_rand\\_num](#) (double mean\_val, double std\_dev\_val)  
*Get a normally distributed random number by applying the Box–Muller method.*
- double [get\\_rand\\_num](#) (int initial\_seed\_val)  
*Generate uniform (0.0, 1.0) random numbers by using the Linear Congruential Generator (LGC) algorithm.*

### 10.57.1 Detailed Description

Generating normally distributed random numbers.

The generation of normally distributed random numbers is implemented by using the Box–Muller method.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.57.2 Function Documentation

#### 10.57.2.1 [get\\_norm\\_distr\\_rand\\_num\(\)](#)

```
double get_norm_distr_rand_num (  
    double mean_val,  
    double std_dev_val )
```

Get a normally distributed random number by applying the Box–Muller method.

#### Parameters

in	<i>mean_val</i>	mean value.
in	<i>std_dev_val</i>	standard deviation.

#### Returns

normally distributed random number.

Definition at line 40 of file `norm_dist_rnd_generator.c`.

References [get\\_rand\\_num\(\)](#), and `PI`.

Referenced by [sim\\_UWB\\_dist\(\)](#).

### 10.57.2.2 `get_rand_num()`

```
double get_rand_num (
    int initial_seed_val )
```

Generate uniform (0.0, 1.0) random numbers by using the Linear Congruential Generator (LGC) algorithm.

#### Note

The LGC is implemented based on the following book: R. Jain, "The Art of Computer Systems Performance Analysis," John Wiley & Sons, 1991.

#### Parameters

in	<code>initial_seed_val</code>	initial seed value.
----	-------------------------------	---------------------

#### Returns

random number between 0.0 and 1.0.

Definition at line 80 of file `norm_dist_rnd_generator.c`.

Referenced by `get_norm_distr_rand_num()`, and `sim_UWB_dist()`.

## 10.58 `norm_dist_rnd_generator.h` File Reference

Generating normally distributed random numbers.

### Macros

- `#define PI` 3.14159265

### Functions

- double `get_norm_distr_rand_num` (double mean, double std\_dev)  
*Get a normally distributed random number by applying the Box–Muller method.*
- double `get_rand_num` (int seed)  
*Generate uniform (0.0, 1.0) random numbers by using the Linear Congruential Generator (LGC) algorithm.*

### 10.58.1 Detailed Description

Generating normally distributed random numbers.

The generation of normally distributed random numbers is implemented by using the Box–Muller method.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.58.2 Macro Definition Documentation

### 10.58.2.1 PI

```
#define PI 3.14159265
```

Pi, the ratio of a circle's circumference to its diameter.

Definition at line 33 of file norm\_dist\_rnd\_generator.h.

## 10.58.3 Function Documentation

### 10.58.3.1 get\_norm\_distr\_rand\_num()

```
double get_norm_distr_rand_num (  
    double mean_val,  
    double std_dev_val )
```

Get a normally distributed random number by applying the Box–Muller method.

#### Parameters

in	<i>mean_val</i>	mean value.
in	<i>std_dev_val</i>	standard deviation.

#### Returns

normally distributed random number.

Definition at line 40 of file norm\_dist\_rnd\_generator.c.

References `get_rand_num()`, and `PI`.

Referenced by `sim_UWB_dist()`.

### 10.58.3.2 get\_rand\_num()

```
double get_rand_num (  
    int initial_seed_val )
```

Generate uniform (0.0, 1.0) random numbers by using the Linear Congruential Generator (LGC) algorithm.

#### Note

The LGC is implemented based on the following book: R. Jain, "The Art of Computer Systems Performance Analysis," John Wiley & Sons, 1991.

**Parameters**

in	<i>initial_seed_val</i>	initial seed value.
----	-------------------------	---------------------

**Returns**

random number between 0.0 and 1.0.

Definition at line 80 of file `norm_dist_rnd_generator.c`.

Referenced by `get_norm_distr_rand_num()`, and `sim_UWB_dist()`.

## 10.59 optimization\_test.c File Reference

Examples of optimization algorithms.

```
#include <stdio.h>
#include <math.h>
#include "matrix.h"
#include "vector.h"
#include "levenberg_marquardt.h"
#include "modified_gauss_newton.h"
```

**Functions**

- void [optimization\\_get\\_J](#) ([vector\\_t](#) x0\_vec[], [matrix\\_t](#) J[][3])  
*Calculate the Jacobian matrix of the function [optimization\\_get\\_f\\_error](#).*
- void [optimization\\_get\\_f\\_error](#) ([vector\\_t](#) x0\_vec[], [vector\\_t](#) measured\_data\_vec[], [vector\\_t](#) f\_vec[])  
*Calculate the error vector of the approximation.*
- void [optimization\\_test](#) (void)  
*Examples of optimization algorithms using the LVM and GN algorithms.*
- void [optimization\\_get\\_exp\\_f](#) ([vector\\_t](#) x\_vec[], [vector\\_t](#) data\_vec[], [vector\\_t](#) f\_vec[])  
*Calculate the error vector using exponential data.*
- void [optimization\\_get\\_exp\\_Jacobian](#) ([vector\\_t](#) x\_vec[], [matrix\\_t](#) J[][2])  
*Calculate the Jacobian matrix using exponential data.*
- void [optimization\\_exponential\\_data\\_test](#) (void)  
*Examples of optimization algorithms using exponential data.*
- void [optimization\\_get\\_sin\\_f](#) ([vector\\_t](#) x\_vec[], [vector\\_t](#) data\_vec[], [vector\\_t](#) f\_vec[])  
*Calculate the error vector using sinusoidal data.*
- void [optimization\\_get\\_sin\\_Jacobian](#) ([vector\\_t](#) x\_vec[], [matrix\\_t](#) J[][4])  
*Calculate the Jacobian matrix using sinusoidal data.*
- void [optimization\\_sinusoidal\\_data\\_test](#) (void)  
*Examples of optimization algorithms using sinusoidal data.*

## 10.59.1 Detailed Description

Examples of optimization algorithms.

Optimization algorithms examples (see the [modified GN](#) and [LVM](#) optimization methods).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.59.2 Function Documentation

### 10.59.2.1 optimization\_exponential\_data\_test()

```
void optimization_exponential_data_test (
    void )
```

Examples of optimization algorithms using exponential data.

The model function is:  $g(\vec{x}, t) = x_1 e^{x_2 t}$ , where  $\vec{x} = [x_1, x_2]^T$  and  $\vec{x}_0 = [6, .3]$  is the initial guess. The data set is  $d(t_i, y_i)$ , whereby  $t_i$  is equal to  $\{1, \dots, 8\}$  and  $y_i$  is equal to  $\{8.3, 11.0, 14.7, 19.7, 26.7, 35.2, 44.4, 55.9\}$ .

Definition at line 264 of file optimization\_test.c.

References [matrix\\_t](#), [modified\\_gauss\\_newton\(\)](#), [opt\\_levenberg\\_marquardt\(\)](#), [optimization\\_get\\_exp\\_f\(\)](#), [optimization\\_get\\_exp\\_Jacobian\(\)](#), [vector\\_clear\(\)](#), and [vector\\_t](#).

### 10.59.2.2 optimization\_get\_exp\_f()

```
void optimization_get_exp_f (
    vector_t x_vec[],
    vector_t data_vec[],
    vector_t f_vec[] )
```

Calculate the error vector using exponential data.

The error function is:  $\vec{f}(x_1, x_2) = [x_1 e^{x_2} - y_1, \dots, x_1 e^{8x_2} - y_8]^T$ ,

Parameters

in	<a href="#">x_vec[]</a>	start vector.
in	<a href="#">data_vec[]</a>	data vector.
out	<a href="#">f_vec[]</a>	calculated error vector.

Definition at line 215 of file optimization\_test.c.

References vector\_t.

Referenced by optimization\_exponential\_data\_test().

### 10.59.2.3 optimization\_get\_exp\_Jacobian()

```
void optimization_get_exp_Jacobian (
    vector_t x_vec[],
    matrix_t J[][2] )
```

Calculate the Jacobian matrix using exponential data.

The Jacobian matrix is:  $J_f = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \\ \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} \end{bmatrix} = \begin{bmatrix} e^{x_2} & e^{x_2} x_1 \\ e^{2x_2} & 2e^{2x_2} x_1 \\ \vdots & \vdots \\ e^{8x_2} & 8e^{8x_2} x_1 \end{bmatrix}.$

#### Parameters

in	<i>x_vec[]</i>	start vector.
in	<i>J[]</i>	Jacobian matrix.

Definition at line 251 of file optimization\_test.c.

References vector\_t.

Referenced by optimization\_exponential\_data\_test().

### 10.59.2.4 optimization\_get\_f\_error()

```
void optimization_get_f_error (
    vector_t x0_vec[],
    vector_t measured_data_vec[],
    vector_t f_vec[] )
```

Calculate the error vector of the approximation.

#### Parameters

in	<i>x0_vec[]</i>	start values.
in	<i>measured_data_vec[]</i>	measured data vector.
out	<i>f_vec[]</i>	calculated error vector.

Definition at line 70 of file optimization\_test.c.

References matrix\_t.

Referenced by optimization\_test().

### 10.59.2.5 optimization\_get\_J()

```
void optimization_get_J (
    vector_t x0_vec[],
    matrix_t J[][3] )
```

Calculate the Jacobian matrix of the function [optimization\\_get\\_f\\_error](#).

#### Parameters

in	<i>x0_vec[]</i>	start values.
out	<i>J[][]</i>	calculated Jacobian matrix.

Definition at line 39 of file optimization\_test.c.

References [matrix\\_t](#).

Referenced by optimization\_test().

### 10.59.2.6 optimization\_get\_sin\_f()

```
void optimization_get_sin_f (
    vector_t x_vec[],
    vector_t data_vec[],
    vector_t f_vec[] )
```

Calculate the error vector using sinusoidal data.

The error function is:  $\vec{f}(x_1, x_2, x_3, x_4) = \begin{bmatrix} x_1 \sin(x_2 + x_3) + x_4 - y_1 \\ \vdots \\ x_1 \sin(12x_2 + x_3) + x_4 - y_{12} \end{bmatrix}$ .

#### Parameters

in	<i>x_vec[]</i>	start vector.
in	<i>data_vec[]</i>	data vector.
out	<i>f_vec[]</i>	calculated error vector.

Definition at line 325 of file optimization\_test.c.

References [vector\\_t](#).

Referenced by optimization\_sinusoidal\_data\_test().

### 10.59.2.7 optimization\_get\_sin\_Jacobian()

```
void optimization_get_sin_Jacobian (
    vector_t x_vec[],
    matrix_t J[] [4] )
```

Calculate the Jacobian matrix using sinusoidal data.

The Jacobian matrix is  $J_f =$

$$\begin{bmatrix} \sin(x_2 + x_3) & x_1 \cos(x_2 + x_3) & x_1 \cos(x_2 + x_3) \\ \sin(2x_2 + x_3) & 2x_1 \cos(2x_2 + x_3) & x_1 \cos(2x_2 + x_3) \\ \vdots & \vdots & \vdots \\ \sin(12x_2 + x_3) & 12x_1 \cos(12x_2 + x_3) & x_1 \cos(12x_2 + x_3) \end{bmatrix}.$$

#### Parameters

in	$x\_vec[]$	start vector.
in	$J[]$	Jacobian matrix.

Definition at line 360 of file optimization\_test.c.

References vector\_t.

Referenced by optimization\_sinusoidal\_data\_test().

### 10.59.2.8 optimization\_sinusoidal\_data\_test()

```
void optimization_sinusoidal_data_test (
    void )
```

Examples of optimization algorithms using sinusoidal data.

The model function is:  $g(\vec{x}, t) = x_1 \sin(x_2 t + x_3) + x_4$ , whereby  $\vec{x} = [x_1, x_2, x_3, x_4]^T$  and  $\vec{x}_0 = [17, 0.5, 10.5, 77]$  is the initial guess. The set of data points is  $d(t_i, y_i)$ , where  $t_i$  is equal to  $\{1, \dots, 12\}$  and  $y_i$  is equal to  $\{61, 65, 72, 78, 85, 90, 92, 92, 88, 81, 72, 63\}$ .

Definition at line 376 of file optimization\_test.c.

References matrix\_t, modified\_gauss\_newton(), opt\_levenberg\_marquardt(), optimization\_get\_sin\_f(), optimization\_get\_sin\_Jacobian(), vector\_clear(), and vector\_t.

## 10.60 optimization\_test.h File Reference

Examples of optimization algorithms.

## Functions

- void [optimization\\_test](#) (void)  
*Examples of optimization algorithms using the LVM and GN algorithms.*
- void [optimization\\_exponential\\_data\\_test](#) (void)  
*Examples of optimization algorithms using exponential data.*
- void [optimization\\_sinusoidal\\_data\\_test](#) (void)  
*Examples of optimization algorithms using sinusoidal data.*

### 10.60.1 Detailed Description

Examples of optimization algorithms.

Optimization algorithms examples (see the [modified GN](#) and [LVM](#) optimization methods).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.60.2 Function Documentation

#### 10.60.2.1 optimization\_exponential\_data\_test()

```
void optimization_exponential_data_test (
    void )
```

Examples of optimization algorithms using exponential data.

The model function is:  $g(\vec{x}, t) = x_1 e^{x_2 t}$ , where  $\vec{x} = [x_1, x_2]^T$  and  $\vec{x}_0 = [6, .3]$  is the initial guess. The data set is  $d(t_i, y_i)$ , whereby  $t_i$  is equal to  $\{1, \dots, 8\}$  and  $y_i$  is equal to  $\{8.3, 11.0, 14.7, 19.7, 26.7, 35.2, 44.4, 55.9\}$ .

Definition at line 264 of file optimization\_test.c.

References [matrix\\_t](#), [modified\\_gauss\\_newton\(\)](#), [opt\\_levenberg\\_marquardt\(\)](#), [optimization\\_get\\_exp\\_f\(\)](#), [optimization\\_get\\_exp\\_Jacobian\(\)](#), [vector\\_clear\(\)](#), and [vector\\_t](#).

#### 10.60.2.2 optimization\_sinusoidal\_data\_test()

```
void optimization_sinusoidal_data_test (
    void )
```

Examples of optimization algorithms using sinusoidal data.

The model function is:  $g(\vec{x}, t) = x_1 \sin(x_2 t + x_3) + x_4$ , whereby  $\vec{x} = [x_1, x_2, x_3, x_4]^T$  and  $\vec{x}_0 = [17, 0.5, 10.5, 77]$  is the initial guess. The set of data points is  $d(t_i, y_i)$ , where  $t_i$  is equal to  $\{1, \dots, 12\}$  and  $y_i$  is equal to  $\{61, 65, 72, 78, 85, 90, 92, 92, 88, 81, 72, 63\}$ .

Definition at line 376 of file optimization\_test.c.

References [matrix\\_t](#), [modified\\_gauss\\_newton\(\)](#), [opt\\_levenberg\\_marquardt\(\)](#), [optimization\\_get\\_sin\\_f\(\)](#), [optimization\\_get\\_sin\\_Jacobian\(\)](#), [vector\\_clear\(\)](#), and [vector\\_t](#).

## 10.61 pos\_algos\_common\_test.c File Reference

Examples of common algorithms of localization systems.

```
#include <stdio.h>
#include "DOP.h"
#include "matrix.h"
#include "trilateration.h"
```

### Functions

- void [pos\\_algos\\_common\\_test](#) (void)  
*Example of common localization algorithms.*

#### 10.61.1 Detailed Description

Examples of common algorithms of localization systems.

Common algorithms examples of positioning systems (see the [positioning algorithms common](#) module).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.62 pos\_algos\_common\_test.h File Reference

Examples of common algorithms of localization systems.

### Functions

- void [pos\\_algos\\_common\\_test](#) (void)  
*Example of common localization algorithms.*

#### 10.62.1 Detailed Description

Examples of common algorithms of localization systems.

Common algorithms examples of positioning systems (see the [positioning algorithms common](#) module).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.63 position\_optimization\_test.c File Reference

Examples of optimization algorithms for localization systems.

```
#include <stdio.h>
#include <stdint.h>
#include "loc_levenberg_marquardt.h"
#include "loc_gauss_newton.h"
#include "dist_based_fi.h"
#include "dist_based_position.h"
#include "dist_based_jacobian.h"
#include "vector.h"
```

### Functions

- void [position\\_optimization\\_test](#) (void)  
*Examples of optimization algorithms of a localization system.*

#### 10.63.1 Detailed Description

Examples of optimization algorithms for localization systems.

Optimization algorithms examples for localization systems. (see the [modified GN](#) and [LVM](#) optimization methods).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
Naouar Guerchali  
Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

## 10.64 position\_optimization\_test.h File Reference

Examples of optimization algorithms for localization systems.

### Functions

- void [position\\_optimization\\_test](#) (void)  
*Examples of optimization algorithms of a localization system.*

#### 10.64.1 Detailed Description

Examples of optimization algorithms for localization systems.

Optimization algorithms examples for localization systems. (see the [modified GN](#) and [LVM](#) optimization methods).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
Naouar Guerchali  
Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

## 10.65 pseudo\_inverse.h File Reference

Compute the pseudo-inverse of a matrix.

### Enumerations

- enum `ALGORITHM` { `Moore_Penrose`, `Householder`, `Givens`, `Gauss` }

*Possible algorithms to compute the pseudo-inverse matrix.*

### 10.65.1 Detailed Description

Compute the pseudo-inverse of a matrix.

The pseudo-inverse matrix can be computed using the Singular Value Decomposition (SVD), Householder, or Givens algorithms.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.65.2 Enumeration Type Documentation

#### 10.65.2.1 ALGORITHM

enum `ALGORITHM`

Possible algorithms to compute the pseudo-inverse matrix.

Enumerator

Moore_Penrose	Moore–Penrose algorithm.
Householder	Householder algorithm.
Givens	Givens algorithm.
Gauss	Gaussian elimination with pivoting algorithm.

Definition at line 30 of file `pseudo_inverse.h`.

## 10.66 qr\_common.c File Reference

Common definitions and implementations for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using.

```
#include <inttypes.h>
#include <stdio.h>
#include <string.h>
#include "matrix.h"
```

## Functions

- void [qr\\_common\\_backward\\_subst](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) U[ ][n], [matrix\\_t](#) b[m], [matrix\\_t](#) x\_sol[m])  
*Implements the backward substitution algorithm.*
- void [qr\\_common\\_get\\_reduced\\_QR](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) Q[m][m], [matrix\\_t](#) R[m][n], [matrix\\_t](#) reduc\_Q[m][n], [matrix\\_t](#) reduc\_R[n][n])  
*Compute the reduced form of the QR-decomposition algorithm.*

### 10.66.1 Detailed Description

Common definitions and implementations for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.66.2 Function Documentation

#### 10.66.2.1 [qr\\_common\\_backward\\_subst\(\)](#)

```
void qr_common_backward_subst (
    uint8_t m,
    uint8_t n,
    matrix\_t U[ ][n],
    matrix\_t b[m],
    matrix\_t x_sol[m] )
```

Implements the backward substitution algorithm.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>U[ ][ ]</i>	pointer to the matrix U.
in	<i>b[ ]</i>	pointer to the vector b.
out	<i>x_sol[ ]</i>	pointer to the solution of the substitution algorithm.

Definition at line 28 of file qr\_common.c.

References [matrix\\_t](#).

Referenced by `solve_givens()`, `solve_householder()`, and `solve_lu_decomp()`.

### 10.66.2.2 `qr_common_get_reduced_QR()`

```
void qr_common_get_reduced_QR (
    uint8_t m,
    uint8_t n,
    matrix_t Q[m][m],
    matrix_t R[m][n],
    matrix_t red_Q[m][n],
    matrix_t red_R[n][n] )
```

Compute the reduced form of the QR-decomposition algorithm.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>Q</i> [][]	pointer to the matrix Q.
in	<i>R</i> [][]	pointer to the matrix R.
out	<i>red_Q</i> [][]	pointer to the reduced matrix Q.
out	<i>red_R</i> [][]	pointer to the reduced matrix R.

Definition at line 59 of file `qr_common.c`.

References `matrix_part_copy()`.

## 10.67 `qr_common.h` File Reference

Common definitions and implementations for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using.

```
#include <inttypes.h>
#include "matrix.h"
```

### Enumerations

- enum `QR_ALGORITHM` { `QR_Householder`, `QR_Givens` }  
*Possible algorithms to compute the QR-decomposition of a matrix.*

### Functions

- void `qr_common_backward_subst` (uint8\_t m, uint8\_t n, matrix\_t U[][n], matrix\_t b[m], matrix\_t x\_sol[m])  
*Implements the backward substitution algorithm.*
- void `qr_common_get_reduced_QR` (uint8\_t m, uint8\_t n, matrix\_t Q[m][m], matrix\_t R[m][n], matrix\_t red\_Q[m][n], matrix\_t red\_R[n][n])  
*Compute the reduced form of the QR-decomposition algorithm.*

## 10.67.1 Detailed Description

Common definitions and implementations for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using.

### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.67.2 Function Documentation

### 10.67.2.1 qr\_common\_backward\_subst()

```
void qr_common_backward_subst (
    uint8_t m,
    uint8_t n,
    matrix_t U[ ][n],
    matrix_t b[m],
    matrix_t x_sol[m] )
```

Implements the backward substitution algorithm.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in	<i>U[ ][ ]</i>	pointer to the matrix U.
in	<i>b[ ]</i>	pointer to the vector b.
out	<i>x_sol[ ]</i>	pointer to the solution of the substitution algorithm.

Definition at line 28 of file qr\_common.c.

References `matrix_t`.

Referenced by `solve_givens()`, `solve_householder()`, and `solve_lu_decomp()`.

### 10.67.2.2 qr\_common\_get\_reduced\_QR()

```
void qr_common_get_reduced_QR (
    uint8_t m,
    uint8_t n,
    matrix_t Q[m][m],
    matrix_t R[m][n],
    matrix_t red_Q[m][n],
    matrix_t red_R[n][n] )
```

Compute the reduced form of the QR-decomposition algorithm.

**Parameters**

in	$m$	row number of the matrix.
in	$n$	column number of the matrix.
in	$Q[[]]$	pointer to the matrix Q.
in	$R[[]]$	pointer to the matrix R.
out	$red\_Q[[]]$	pointer to the reduced matrix Q.
out	$red\_R[[]]$	pointer to the reduced matrix R.

Definition at line 59 of file qr\_common.c.

References `matrix_part_copy()`.

## 10.68 qr\_givens.c File Reference

Givens algorithm for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using Givens rotations.  $A = QR$ , where Q is an  $(m \times n)$ -matrix with orthonormal columns and R is an  $(n \times n)$  upper triangular matrix.

```
#include <stdio.h>
#include <math.h>
#include <float.h>
#include "matrix.h"
#include "qr_givens.h"
```

### Functions

- `int8_t qr_givens_decomp` (`uint8_t m`, `uint8_t n`, `matrix_t A[ ][n]`, `uint8_t q_col_num`, `matrix_t Q[ ][q_col_num]`, `bool reduced`)  
*Computes the QR decomposition of the matrix A by using the Givens algorithm.*
- `void qr_givens_get_params` (`matrix_t xjj`, `matrix_t xij`, `matrix_t c_s_t_r_vec[ ]`)  
*Compute the Givens parameters.*

### 10.68.1 Detailed Description

Givens algorithm for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using Givens rotations.  $A = QR$ , where Q is an  $(m \times n)$ -matrix with orthonormal columns and R is an  $(n \times n)$  upper triangular matrix.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.68.2 Function Documentation

### 10.68.2.1 qr\_givens\_decomp()

```
int8_t qr_givens_decomp (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    uint8_t q_col_num,
    matrix_t Q[ ][q_col_num],
    bool reduced )
```

Computes the QR decomposition of the matrix A by using the Givens algorithm.

Gets a QR decomposition of an m-by-n matrix A such that  $A = Q \cdot R$ . The compact as well as the full decomposition of the matrix can be computed.

#### Note

R is stored in the matrix A.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in, out	<i>A[ ][ ]</i>	pointer to the matrices A and R.
in	<i>q_col_num</i>	column number of the matrix Q.
out	<i>Q[ ][ ]</i>	pointer to the Q matrix.
in	<i>reduced</i>	computes the compact form of the QR decomposition if true, otherwise the full version.

#### Returns

- 1, if computing the QR decomposition is successful.
- 1, if computing the QR decomposition is not successful.

Definition at line 33 of file qr\_givens.c.

References `matrix_clear()`, `matrix_get_upp_triangular()`, `matrix_part_copy()`, `matrix_t`, and `qr_givens_get_params()`.

Referenced by `givens_test()`, and `solve_givens()`.

### 10.68.2.2 qr\_givens\_get\_params()

```
void qr_givens_get_params (
    matrix_t xjj,
    matrix_t xij,
    matrix_t c_s_t_r_vec[ ] )
```

Compute the Givens parameters.

The computation of the parameters c, s, t, and r can have problems with overflow or underflow, therefore this algorithm employs a normalization procedure. The Givens parameters c, s, t and r are saved in a vector.

**Parameters**

in	<i>xij</i>	value at the diagonal j of the matrix.
in	<i>xij</i>	value at the index j of a column vector i.
out	<i>c_s_t_r_vec[]</i>	pointer to the vector holding the c, s, t and r parameters.

Definition at line 110 of file qr\_givens.c.

References `matrix_t`.

Referenced by `givens_test()`, and `qr_givens_decomp()`.

## 10.69 qr\_givens.h File Reference

Givens algorithm for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using Givens rotations.  $A = QR$ , where Q is an  $(m \times n)$ -matrix with orthonormal columns and R is an  $(n \times n)$  upper triangular matrix.

```
#include <inttypes.h>
#include "matrix.h"
```

### Functions

- `int8_t qr_givens_decomp` (`uint8_t m`, `uint8_t n`, `matrix_t A[][n]`, `uint8_t q_col_num`, `matrix_t Q[][q_col_num]`, `bool reduced`)  
*Computes the QR decomposition of the matrix A by using the Givens algorithm.*
- `void qr_givens_get_params` (`matrix_t xjj`, `matrix_t xij`, `matrix_t c_s_t_r_vec[]`)  
*Compute the Givens parameters.*

### 10.69.1 Detailed Description

Givens algorithm for the QR-decomposition. Provide necessary methods to construct Q- and R- matrices using Givens rotations.  $A = QR$ , where Q is an  $(m \times n)$ -matrix with orthonormal columns and R is an  $(n \times n)$  upper triangular matrix.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.69.2 Function Documentation

### 10.69.2.1 qr\_givens\_decomp()

```
int8_t qr_givens_decomp (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    uint8_t q_col_num,
    matrix_t Q[ ][q_col_num],
    bool reduced )
```

Computes the QR decomposition of the matrix A by using the Givens algorithm.

Gets a QR decomposition of an m-by-n matrix A such that  $A = Q \cdot R$ . The compact as well as the full decomposition of the matrix can be computed.

#### Note

R is stored in the matrix A.

#### Parameters

in	<i>m</i>	row number of the matrix.
in	<i>n</i>	column number of the matrix.
in, out	<i>A[ ][ ]</i>	pointer to the matrices A and R.
in	<i>q_col_num</i>	column number of the matrix Q.
out	<i>Q[ ][ ]</i>	pointer to the Q matrix.
in	<i>reduced</i>	computes the compact form of the QR decomposition if true, otherwise the full version.

#### Returns

- 1, if computing the QR decomposition is successful.
- 1, if computing the QR decomposition is not successful.

Definition at line 33 of file qr\_givens.c.

References `matrix_clear()`, `matrix_get_upp_triangular()`, `matrix_part_copy()`, `matrix_t`, and `qr_givens_get_params()`.

Referenced by `givens_test()`, and `solve_givens()`.

### 10.69.2.2 qr\_givens\_get\_params()

```
void qr_givens_get_params (
    matrix_t xjj,
    matrix_t xij,
    matrix_t c_s_t_r_vec[ ] )
```

Compute the Givens parameters.

The computation of the parameters c, s, t, and r can have problems with overflow or underflow, therefore this algorithm employs a normalization procedure. The Givens parameters c, s, t and r are saved in a vector.

**Parameters**

in	<i>x<sub>jj</sub></i>	value at the diagonal j of the matrix.
in	<i>x<sub>ij</sub></i>	value at the index j of a column vector i.
out	<i>c_s_t_r_vec[]</i>	pointer to the vector holding the c, s, t and r parameters.

Definition at line 110 of file qr\_givens.c.

References `matrix_t`.

Referenced by `givens_test()`, and `qr_givens_decomp()`.

## 10.70 qr\_householder.c File Reference

Householder algorithm for the QR-decomposition.

```
#include <stdio.h>
#include <stdbool.h>
#include <math.h>
#include "matrix.h"
#include "utils.h"
```

### Functions

- `int8_t qr_householder_decomp` (uint8\_t m, uint8\_t n, `matrix_t` A[][n], uint8\_t q\_col\_num, `matrix_t` Q[][q\_col\_num], bool reduced)

*Computes the QR decomposition of the matrix A by using the Householder algorithm.*

#### 10.70.1 Detailed Description

Householder algorithm for the QR-decomposition.

Provide necessary methods to construct Q- and R- matrices using Householder reflections.  $A = QR$ , where Q is an  $(m \times n)$ -matrix with orthonormal columns and R is an  $(n \times n)$  upper triangular matrix.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

#### 10.70.2 Function Documentation

### 10.70.2.1 qr\_householder\_decomp()

```
int8_t qr_householder_decomp (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    uint8_t q_col_num,
    matrix_t Q[ ][q_col_num],
    bool reduced )
```

Computes the QR decomposition of the matrix A by using the Householder algorithm.

#### Note

R is stored in the matrix A.

#### Parameters

in	<i>m</i>	row number of the matrix to decompose in QR.
in	<i>n</i>	column number of the matrix to decompose in QR.
in, out	<i>A[ ][ ]</i>	pointer to the matrices A and R.
in	<i>q_col_num</i>	column number of the matrix Q.
in, out	<i>Q[ ][ ]</i>	pointer to the matrix Q.
in	<i>reduced</i>	computes the compact form of the QR decomposition if true, otherwise the full version.

#### Returns

- 1, if computing the QR decomposition is successful.
- 1, if computing the QR decomposition is not successful.

Definition at line 33 of file qr\_householder.c.

References `matrix_t`, and `utils_get_save_square_root()`.

Referenced by `householder_test()`, and `solve_householder()`.

## 10.71 qr\_householder.h File Reference

Householder algorithm for the QR-decomposition.

```
#include <inttypes.h>
#include "matrix.h"
```

### Functions

- `int8_t qr_householder_decomp (uint8_t m, uint8_t n, matrix_t A[ ][n], uint8_t q_col_num, matrix_t Q[ ][q_col_num], bool reduced)`  
*Computes the QR decomposition of the matrix A by using the Householder algorithm.*

### 10.71.1 Detailed Description

Householder algorithm for the QR-decomposition.

Provide necessary methods to construct Q- and R- matrices using Householder reflections.  $A = QR$ , where Q is an  $(m \times n)$ -matrix with orthonormal columns and R is an  $(n \times n)$  upper triangular matrix.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.71.2 Function Documentation

#### 10.71.2.1 `qr_householder_decomp()`

```
int8_t qr_householder_decomp (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    uint8_t q_col_num,
    matrix_t Q[ ][q_col_num],
    bool reduced )
```

Computes the QR decomposition of the matrix A by using the Householder algorithm.

#### Note

R is stored in the matrix A.

#### Parameters

in	<i>m</i>	row number of the matrix to decompose in QR.
in	<i>n</i>	column number of the matrix to decompose in QR.
in, out	<i>A[ ][ ]</i>	pointer to the matrices A and R.
in	<i>q_col_num</i>	column number of the matrix Q.
in, out	<i>Q[ ][ ]</i>	pointer to the matrix Q.
in	<i>reduced</i>	computes the compact form of the QR decomposition if true, otherwise the full version.

#### Returns

- 1, if computing the QR decomposition is successful.
- 1, if computing the QR decomposition is not successful.

Definition at line 33 of file `qr_householder.c`.

References `matrix_t`, and `utils_get_save_square_root()`.

Referenced by `householder_test()`, and `solve_householder()`.

## 10.72 qr\_pinv\_test.c File Reference

Examples of the QR-based pseudo-inverse algorithm.

```
#include <qr_pseudo_inverse.h>
#include <stdio.h>
#include "matrix.h"
#include "qr_common.h"
```

### Functions

- void [qr\\_pinv\\_test](#) (void)  
*Examples of the QR-based pseudo-inverse algorithm.*

#### 10.72.1 Detailed Description

Examples of the QR-based pseudo-inverse algorithm.

QR-based pseudo-inverse algorithm examples (see the [QR Pseudo-Inverse](#) approach).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.73 qr\_pinv\_test.h File Reference

Examples of the QR-based pseudo-inverse algorithm.

### Functions

- void [qr\\_pinv\\_test](#) (void)  
*Examples of the QR-based pseudo-inverse algorithm.*

#### 10.73.1 Detailed Description

Examples of the QR-based pseudo-inverse algorithm.

QR-based pseudo-inverse algorithm examples (see the [QR Pseudo-Inverse](#) approach).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.74 qr\_pseudo\_inverse.c File Reference

QR decomposition algorithms to compute the pseudo-inverse of a matrix.

```
#include <stdio.h>
#include "matrix.h"
#include "qr_common.h"
#include "qr_givens.h"
#include "qr_householder.h"
#include "pseudo_inverse.h"
```

### Functions

- `int8_t qr_get_pinv (uint8_t m, uint8_t n, matrix\_t A[m][n], matrix\_t pinv_A[n][m], enum QR\_ALGORITHM algo)`  
*Calculate the pseudo inverse of a rectangular matrix using the QR decomposition.*

### 10.74.1 Detailed Description

QR decomposition algorithms to compute the pseudo-inverse of a matrix.

The computation of the pseudo-inverse is implemented using the Householder or the Givens algorithms.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.74.2 Function Documentation

#### 10.74.2.1 qr\_get\_pinv()

```
int8_t qr_get_pinv (
    uint8_t m,
    uint8_t n,
    matrix\_t A[m][n],
    matrix\_t pinv_A[n][m],
    enum QR\_ALGORITHM algo )
```

Calculate the pseudo inverse of a rectangular matrix using the QR decomposition.

The computation of the pseudo inverse is based on the Householder or Givens algorithm.

## Parameters

in	<i>m</i>	row number of the matrix to inverse.
in	<i>n</i>	column number of the matrix to inverse.
in	<i>A</i> [][]	pointer to the matrix A.
out	<i>pinv_A</i> [][]	pointer to the pseudo-inverse matrix.
in	<i>algo</i>	choice between the Householder or Givens algorithms.

## Returns

- 1, if computing the pseudo-inverse matrix is successful.
- 1, if computing the pseudo-inverse matrix is not successful.

Definition at line 32 of file qr\_pseudo\_inverse.c.

Referenced by qr\_pinv\_test().

## 10.75 qr\_pseudo\_inverse.h File Reference

QR decomposition algorithms to compute the pseudo-inverse of a matrix.

```
#include <inttypes.h>
#include "matrix.h"
#include "qr_common.h"
```

### Functions

- `int8_t qr_get_pinv (uint8_t m, uint8_t n, matrix\_t A[m][n], matrix\_t pinv_A[n][m], enum QR\_ALGORITHM algo)`  
*Calculate the pseudo inverse of a rectangular matrix using the QR decomposition.*

### 10.75.1 Detailed Description

QR decomposition algorithms to compute the pseudo-inverse of a matrix.

The computation of the pseudo-inverse is implemented using the Householder or the Givens algorithms.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.75.2 Function Documentation

#### 10.75.2.1 qr\_get\_pinv()

```
int8_t qr_get_pinv (
    uint8_t m,
    uint8_t n,
    matrix\_t A[m][n],
    matrix\_t pinv_A[n][m],
    enum QR\_ALGORITHM algo )
```

Calculate the pseudo inverse of a rectangular matrix using the QR decomposition.

The computation of the pseudo inverse is based on the Householder or Givens algorithm.

**Parameters**

in	<i>m</i>	row number of the matrix to inverse.
in	<i>n</i>	column number of the matrix to inverse.
in	<i>A</i> [][]	pointer to the matrix A.
out	<i>pinv_A</i> [][]	pointer to the pseudo-inverse matrix.
in	<i>algo</i>	choice between the Householder or Givens algorithms.

**Returns**

- 1, if computing the pseudo-inverse matrix is successful.
- 1, if computing the pseudo-inverse matrix is not successful.

Definition at line 32 of file `qr_pseudo_inverse.c`.

Referenced by `qr_pinv_test()`.

## 10.76 shell\_sort.c File Reference

Implement the Shell sort algorithm.

```
#include <stdio.h>
#include "utils.h"
#include "vector.h"
```

**Functions**

- void [int\\_shell\\_sort](#) (int \*array, int length)  
*Sort a data set of integers by using the Shell sort algorithm.*
- void [shell\\_sort](#) (vector\_t \*arr, uint8\_t length)  
*Sort a data set of type utils\_t by using the Shell sort algorithm.*

### 10.76.1 Detailed Description

Implement the Shell sort algorithm.

The Shell sort algorithm is more convenient for devices with limited storage capacity.

**Author**

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.76.2 Function Documentation

#### 10.76.2.1 int\_shell\_sort()

```
void int_shell_sort (
    int * array,
    int length )
```

Sort a data set of integers by using the Shell sort algorithm.

## Parameters

in	<i>array[]</i>	pointer to the data set.
in	<i>length</i>	size of the data set.

Definition at line 28 of file shell\_sort.c.

### 10.76.2.2 shell\_sort()

```
void shell_sort (
    vector_t * arr,
    uint8_t length )
```

Sort a data set of type `utils_t` by using the Shell sort algorithm.

## Parameters

in	<i>arr[]</i>	pointer to the data set.
in	<i>length</i>	size of the data set.

Definition at line 49 of file shell\_sort.c.

References `vector_t`.

Referenced by `recog_mitigate_multipath()`, and `utils_get_median()`.

## 10.77 shell\_sort.h File Reference

Implement the Shell sort algorithm.

```
#include <stdint.h>
#include "vector.h"
```

### Functions

- void `int_shell_sort` (int \*array, int length)  
*Sort a data set of integers by using the Shell sort algorithm.*
- void `shell_sort` (`vector_t` \*arr, uint8\_t length)  
*Sort a data set of type `utils_t` by using the Shell sort algorithm.*

### 10.77.1 Detailed Description

Implement the Shell sort algorithm.

The Shell sort algorithm is more convenient for devices with limited storage capacity.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.77.2 Function Documentation

### 10.77.2.1 `int_shell_sort()`

```
void int_shell_sort (
    int * array,
    int length )
```

Sort a data set of integers by using the Shell sort algorithm.

#### Parameters

in	<i>array[]</i>	pointer to the data set.
in	<i>length</i>	size of the data set.

Definition at line 28 of file `shell_sort.c`.

### 10.77.2.2 `shell_sort()`

```
void shell_sort (
    vector\_t * arr,
    uint8_t length )
```

Sort a data set of type `utils_t` by using the Shell sort algorithm.

#### Parameters

in	<i>arr[]</i>	pointer to the data set.
in	<i>length</i>	size of the data set.

Definition at line 49 of file `shell_sort.c`.

References `vector_t`.

Referenced by `recog_mitigate_multipath()`, and `utils_get_median()`.

## 10.78 `solve.c` File Reference

Enables to solve systems of linear equations  $Ax = b$  for  $x$ .

```
#include <stdio.h>
#include "solve.h"
#include "matrix.h"
#include "qr_givens.h"
```

```
#include "qr_common.h"
#include "lu_decomp.h"
#include "qr_householder.h"
#include "moore_penrose_pseudo_inverse.h"
```

## Functions

- `int8_t solve` (`uint8_t m`, `uint8_t n`, `matrix_t A[][n]`, `matrix_t b[m]`, `matrix_t x_sol[n]`, `enum ALGORITHM algo`)  
Solve an  $(m \times n)$  linear system  $Ax = b$  by using the Moore–Penrose, Householder, or the Givens algorithm.
- `int8_t solve_householder` (`uint8_t m`, `uint8_t n`, `matrix_t A[][n]`, `matrix_t b[m]`, `matrix_t x_sol[n]`)  
Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Householder algorithm.
- `int8_t solve_givens` (`uint8_t m`, `uint8_t n`, `matrix_t A[][n]`, `matrix_t b[m]`, `matrix_t x_sol[n]`)  
Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Givens algorithm.
- `int8_t solve_lu_decomp` (`uint8_t m`, `uint8_t n`, `matrix_t A[][n]`, `matrix_t b[m]`, `matrix_t x_sol[n]`)  
Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Gaussian Elimination with pivoting algorithm.

### 10.78.1 Detailed Description

Enables to solve systems of linear equations  $Ax = b$  for  $x$ .

The user can select various algorithm such as the Moore–Penrose inverse, the Givens or the Householder algorithm for the QR-decomposition to solve the systems of linear equations.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.78.2 Function Documentation

#### 10.78.2.1 solve()

```
int8_t solve (
    uint8_t m,
    uint8_t n,
    matrix_t A[][n],
    matrix_t b[m],
    matrix_t x_sol[n],
    enum ALGORITHM algo )
```

Solve an  $(m \times n)$  linear system  $Ax = b$  by using the Moore–Penrose, Householder, or the Givens algorithm.

**Parameters**

in	<i>m</i>	row number of the matrix A.
in	<i>n</i>	column number of the matrix A.
in	<i>A</i> [][]	pointer to the matrix A.
in	<i>b</i> []	pointer to the vector b.
out	<i>x_sol</i> []	pointer to the solution vector.
in	<i>algo</i>	specifies the algorithm to use (e.g. the Householder method).

**Returns**

- 1, if solving the linear equation system is successful.
- 1, if solving the linear equation system is not successful.

Definition at line 35 of file solve.c.

References Gauss, Givens, Householder, matrix\_mul\_vec(), matrix\_t, Moore\_Penrose, moore\_penrose\_get\_pinv(), solve\_givens(), solve\_householder(), and solve\_lu\_decomp().

Referenced by solve\_big\_matrix\_test(), and solve\_test().

**10.78.2.2 solve\_givens()**

```
int8_t solve_givens (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t b[m],
    matrix_t x_sol[n] )
```

Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Givens algorithm.

**Parameters**

in	<i>m</i>	row number of the matrix A.
in	<i>n</i>	column number of the matrix A.
in	<i>A</i> [][]	pointer to the matrix A.
in	<i>b</i> []	pointer to the vector b.
out	<i>x_sol</i> []	pointer to the solution vector.

**Returns**

- 1, if solving the linear equation system is successful.
- 1, if solving the linear equation system is not successful.
- 1, if the linear system is not solvable.

Definition at line 102 of file solve.c.

References `matrix_t`, `matrix_trans_mul_vec()`, `qr_common_backward_subst()`, and `qr_givens_decomp()`.

Referenced by `solve()`.

### 10.78.2.3 solve\_householder()

```
int8_t solve_householder (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t b[m],
    matrix_t x_sol[n] )
```

Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Householder algorithm.

#### Parameters

in	<i>m</i>	row number of the matrix A.
in	<i>n</i>	column number of the matrix A.
in	<i>A[ ][ ]</i>	pointer to the matrix A.
in	<i>b[ ]</i>	pointer to the vector b.
out	<i>x_sol[ ]</i>	pointer to the solution vector.

#### Returns

- 1, if solving the linear equation system is successful.
- 1, if solving the linear equation system is not successful.

Definition at line 77 of file `solve.c`.

References `matrix_t`, `matrix_trans_mul_vec()`, `qr_common_backward_subst()`, and `qr_householder_decomp()`.

Referenced by `loc_levenberg_marquardt()`, `loc_levenberg_marquardt_correction()`, `opt_levenberg_marquardt()`, `opt_levenberg_marquardt_correction()`, and `solve()`.

### 10.78.2.4 solve\_lu\_decomp()

```
int8_t solve_lu_decomp (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t b[m],
    matrix_t x_sol[n] )
```

Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Gaussian Elimination with pivoting algorithm.

## Parameters

in	$m$	row number of the matrix A.
in	$n$	column number of the matrix A.
in	$A[]$	pointer to the matrix A.
in	$b[]$	pointer to the vector b.
out	$x\_sol[]$	pointer to the solution vector.

## Returns

- 1, if solving the linear equation system is successful.
- 1, if solving the linear equation system is not successful.
- 2, if the linear system is not solvable.

Definition at line 124 of file solve.c.

References `lu_decomp()`, `matrix_get_inv_low_triang()`, `matrix_mul()`, `matrix_mul_vec()`, `matrix_t`, and `qr_common↵_backward_subst()`.

Referenced by `solve()`.

## 10.79 solve.h File Reference

Enables to solve systems of linear equations  $Ax = b$  for  $x$ .

```
#include "matrix.h"
#include "pseudo_inverse.h"
```

### Functions

- `int8_t solve` (`uint8_t m`, `uint8_t n`, `matrix_t A[]`, `matrix_t b[m]`, `matrix_t x_sol[n]`, enum `ALGORITHM algo`)  
Solve an  $(m \times n)$  linear system  $Ax = b$  by using the Moore–Penrose, Householder, or the Givens algorithm.
- `int8_t solve_householder` (`uint8_t m`, `uint8_t n`, `matrix_t A[]`, `matrix_t b[m]`, `matrix_t x_sol[n]`)  
Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Householder algorithm.
- `int8_t solve_givens` (`uint8_t m`, `uint8_t n`, `matrix_t A[]`, `matrix_t b[m]`, `matrix_t x_sol[n]`)  
Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Givens algorithm.
- `int8_t solve_lu_decomp` (`uint8_t m`, `uint8_t n`, `matrix_t A[]`, `matrix_t b[m]`, `matrix_t x_sol[n]`)  
Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Gaussian Elimination with pivoting algorithm.

### 10.79.1 Detailed Description

Enables to solve systems of linear equations  $Ax = b$  for  $x$ .

The user can select various algorithm such as the Moore–Penrose inverse, the Givens or the Householder algorithm for the QR-decomposition. The user can also choose the Gaussian Elimination with pivoting algorithm to solve the systems of linear equations.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.79.2 Function Documentation

### 10.79.2.1 solve()

```
int8_t solve (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t b[m],
    matrix_t x_sol[n],
    enum ALGORITHM algo )
```

Solve an  $(m \times n)$  linear system  $Ax = b$  by using the Moore–Penrose, Householder, or the Givens algorithm.

#### Parameters

in	<i>m</i>	row number of the matrix A.
in	<i>n</i>	column number of the matrix A.
in	<i>A[ ][ ]</i>	pointer to the matrix A.
in	<i>b[ ]</i>	pointer to the vector b.
out	<i>x_sol[ ]</i>	pointer to the solution vector.
in	<i>algo</i>	specifies the algorithm to use (e.g. the Householder method).

#### Returns

- 1, if solving the linear equation system is successful.
- 1, if solving the linear equation system is not successful.

Definition at line 35 of file solve.c.

References Gauss, Givens, Householder, matrix\_mul\_vec(), matrix\_t, Moore\_Penrose, moore\_penrose\_get\_pinv(), solve\_givens(), solve\_householder(), and solve\_lu\_decomp().

Referenced by solve\_big\_matrix\_test(), and solve\_test().

### 10.79.2.2 solve\_givens()

```
int8_t solve_givens (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t b[m],
    matrix_t x_sol[n] )
```

Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Givens algorithm.

**Parameters**

in	<i>m</i>	row number of the matrix A.
in	<i>n</i>	column number of the matrix A.
in	<i>A</i> [][]	pointer to the matrix A.
in	<i>b</i> []	pointer to the vector b.
out	<i>x_sol</i> []	pointer to the solution vector.

**Returns**

- 1, if solving the linear equation system is successful.
- 1, if solving the linear equation system is not successful.
- 1, if the linear system is not solvable.

Definition at line 102 of file solve.c.

References `matrix_t`, `matrix_trans_mul_vec()`, `qr_common_backward_subst()`, and `qr_givens_decomp()`.

Referenced by `solve()`.

**10.79.2.3 solve\_householder()**

```
int8_t solve_householder (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t b[m],
    matrix_t x_sol[n] )
```

Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Householder algorithm.

**Parameters**

in	<i>m</i>	row number of the matrix A.
in	<i>n</i>	column number of the matrix A.
in	<i>A</i> [][]	pointer to the matrix A.
in	<i>b</i> []	pointer to the vector b.
out	<i>x_sol</i> []	pointer to the solution vector.

**Returns**

- 1, if solving the linear equation system is successful.
- 1, if solving the linear equation system is not successful.

Definition at line 77 of file solve.c.

References `matrix_t`, `matrix_trans_mul_vec()`, `qr_common_backward_subst()`, and `qr_householder_decomp()`.

Referenced by `loc_levenberg_marquardt()`, `loc_levenberg_marquardt_correction()`, `opt_levenberg_marquardt()`, `opt_levenberg_marquardt_correction()`, and `solve()`.

### 10.79.2.4 solve\_lu\_decomp()

```
int8_t solve_lu_decomp (
    uint8_t m,
    uint8_t n,
    matrix_t A[ ][n],
    matrix_t b[m],
    matrix_t x_sol[n] )
```

Solve an  $(m \times n)$  linear system  $Ax = b$ , using the Gaussian Elimination with pivoting algorithm.

#### Parameters

in	$m$	row number of the matrix A.
in	$n$	column number of the matrix A.
in	$A[ ][ ]$	pointer to the matrix A.
in	$b[ ]$	pointer to the vector b.
out	$x\_sol[ ]$	pointer to the solution vector.

#### Returns

- 1, if solving the linear equation system is successful.
- 1, if solving the linear equation system is not successful.
- 2, if the linear system is not solvable.

Definition at line 124 of file solve.c.

References `lu_decomp()`, `matrix_get_inv_low_triang()`, `matrix_mul()`, `matrix_mul_vec()`, `matrix_t`, and `qr_common`↔  
`_backward_subst()`.

Referenced by `solve()`.

## 10.80 solve\_test.c File Reference

Examples of solving linear equation systems.

```
#include <stdio.h>
#include "solve.h"
#include "matrix.h"
#include "vector.h"
```

### Functions

- void `solve_test` (void)  
*Examples of solving linear equation systems.*
- void `solve_big_matrix_test` (void)  
*Example of solving an (10,5) linear equation system.*

### 10.80.1 Detailed Description

Examples of solving linear equation systems.

Solving linear equation systems examples (see [solve](#) functions).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.81 solve\_test.h File Reference

Examples of solving linear equation systems.

### Functions

- void [solve\\_test](#) (void)  
*Examples of solving linear equation systems.*
- void [solve\\_big\\_matrix\\_test](#) (void)  
*Example of solving an (10,5) linear equation system.*

### 10.81.1 Detailed Description

Examples of solving linear equation systems.

Solving linear equation systems examples (see [solve](#) functions).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.82 svd.c File Reference

Algorithm for the Singular Value Decomposition (SVD). Provide necessary methods to compute the compact SVD of a matrix.  $A = U \cdot S \cdot V$ , where  $U$  is a  $(m \times l)$  orthogonal matrix,  $S$  is a  $(l \times l)$  diagonal matrix,  $V$  is a  $(l \times n)$  orthogonal matrix, and  $l = \min(m, n)$ . The SVD is computed by using the Golub–Kahan–Reinsch algorithm that works in two phases: bidiagonalization and a reduction to the diagonal form phase.

```
#include <stdbool.h>
#include <inttypes.h>
#include <math.h>
#include <string.h>
#include <stdio.h>
#include "svd.h"
#include "matrix.h"
#include "vector.h"
```

## Functions

- void `svd_get_U_dim` (uint8\_t m, uint8\_t n, `matrix_dim_t` \*u\_dim)  
*Calculate the dimension of the matrix U.*
- void `svd_get_S_dim` (uint8\_t m, uint8\_t n, `matrix_dim_t` \*s\_dim)  
*Calculate the dimension of the matrix S.*
- void `svd_get_V_dim` (uint8\_t m, uint8\_t n, `matrix_dim_t` \*v\_dim)  
*Calculate the dimension of the matrix V.*
- uint8\_t `svd_get_single_values_num` (uint8\_t m, uint8\_t n)  
*Calculate the number of the singular values.*
- void `svd` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], uint8\_t u\_m, uint8\_t u\_n, `matrix_t` U[u\_m][u\_n], `matrix_t` S[u\_n][n], `matrix_t` V[n][n], uint8\_t sing\_vec\_length, `matrix_t` singl\_values\_vec[])  
*Compute the Singular-Value Decomposition (SVD) of a matrix.*
- void `svd_get_reciproc_singular_values` (uint8\_t m, uint8\_t n, uint8\_t length, `matrix_t` singl\_values\_arr[], `matrix_t` recip\_singl\_values\_arr[])  
*Compute the reciprocal singular values.*
- void `svd_compute_print_U_S_V_s` (uint8\_t m, uint8\_t n, `matrix_t` matrix\_arr[m][n], uint8\_t i)  
*Compute and print the SVD of a matrix.*

### 10.82.1 Detailed Description

Algorithm for the Singular Value Decomposition (SVD). Provide necessary methods to compute the compact SVD of a matrix.  $A = U \cdot S \cdot V$ , where U is a (m x l) orthogonal matrix, S is a (l x l) diagonal matrix, V is a (l x n) orthogonal matrix, and  $l = \min(m, n)$ . The SVD is computed by using the Golub–Kahan–Reinsch algorithm that works in two phases: bidiagonalization and a reduction to the diagonal form phase.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.82.2 Function Documentation

#### 10.82.2.1 `svd()`

```
void svd (
    uint8_t m,
    uint8_t n,
    matrix_t A[m][n],
    uint8_t u_m,
    uint8_t u_n,
    matrix_t U[u_m][u_n],
    matrix_t S[u_n][n],
    matrix_t V[n][n],
    uint8_t sing_vec_length,
    matrix_t singl_values_vec[] )
```

Compute the Singular-Value Decomposition (SVD) of a matrix.

Matrix A is transformed to  $A = U \cdot S \cdot V$ , where U and V are unitary matrices, and S is a diagonal matrix.

## Parameters

in	<i>m</i>	row number of the matrix A.
in	<i>n</i>	column number of the matrix A.
in, out	<i>A</i> [][]	pointer to the matrix A.
in	<i>u_m</i>	row number of the matrix U.
in	<i>u_n</i>	column number of the matrix U.
in, out	<i>U</i> [][]	pointer to the matrix U.
in, out	<i>S</i> [][]	pointer to the matrix S.
in, out	<i>V</i> [][]	pointer to the matrix V.
in	<i>sing_vec_length</i>	length of the singular vector.
in, out	<i>singl_values_vec</i> []	pointer to the vector saving the singular values.

Definition at line 119 of file svd.c.

Referenced by `matrix_get_two_norm()`, `svd_compute_print_U_S_V_s()`, and `trilateration_get_rank_and_homogeneous_solution()`.

### 10.82.2.2 svd\_compute\_print\_U\_S\_V\_s()

```
void svd_compute_print_U_S_V_s (
    uint8_t m,
    uint8_t n,
    matrix_t matrix_arr[m][n],
    uint8_t i )
```

Compute and print the SVD of a matrix.

## Parameters

in	<i>m</i>	row number of the matrix to transform in SVD.
in	<i>n</i>	column number of the matrix to transform in SVD.
in	<i>matrix_arr</i> [][]	pointer to the matrix.
in	<i>i</i>	label.

Definition at line 766 of file svd.c.

References `matrix_dim_t::col_num`, `matrix_print()`, `matrix_t`, `matrix_dim_t::row_num`, `svd()`, `svd_get_single_↵`  
`values_num()`, and `svd_get_U_dim()`.

Referenced by `svd_test()`.

### 10.82.2.3 svd\_get\_reciproc\_singular\_values()

```
void svd_get_reciproc_singular_values (
    uint8_t m,
    uint8_t n,
    uint8_t length,
    matrix_t singl_values_arr[],
    matrix_t recip_singl_values_arr[] )
```

Compute the reciprocal singular values.

This method is based on the singular values.

## Parameters

in	<i>m</i>	row number of the matrix to transform in SVD.
in	<i>n</i>	column number of the matrix to transform in SVD.
in	<i>length</i>	length of the array of single values.
in	<i>singl_values_arr</i>	pointer to the array of single values.
out	<i>recip_singl_values_arr</i>	pointer to the array of the reciprocal singular values.

Definition at line 747 of file svd.c.

References `matrix_t`.

### 10.82.2.4 svd\_get\_S\_dim()

```
void svd_get_S_dim (
    uint8_t m,
    uint8_t n,
    matrix_dim_t * s_dim )
```

Calculate the dimension of the matrix S.

**Parameters**

in	<i>m</i>	row number of the matrix to decompose.
in	<i>n</i>	column number of the matrix to decompose.
out	<i>s_dim</i>	pointer to the dimension struct.

Definition at line 102 of file svd.c.

References `matrix_dim_t::row_num`.

**10.82.2.5 svd\_get\_single\_values\_num()**

```
uint8_t svd_get_single_values_num (
    uint8_t m,
    uint8_t n )
```

Calculate the number of the singular values.

**Parameters**

in	<i>m</i>	row number of the matrix to decompose.
in	<i>n</i>	column number of the matrix to decompose.

**Returns**

the number of the singular values.

Definition at line 114 of file svd.c.

Referenced by `matrix_get_two_norm()`, `moore_penrose_get_pinv()`, `svd_compute_print_U_S_V_s()`, and `trilateration_get_rank_and_homogeneous_solution()`.

**10.82.2.6 svd\_get\_U\_dim()**

```
void svd_get_U_dim (
    uint8_t m,
    uint8_t n,
    matrix_dim_t * u_dim )
```

Calculate the dimension of the matrix U.

**Parameters**

in	<i>m</i>	row number of the matrix to decompose.
in	<i>n</i>	column number of the matrix to decompose.
out	<i>u_dim</i>	pointer to the dimension struct.

Definition at line 96 of file svd.c.

References `matrix_dim_t::col_num`, and `matrix_dim_t::row_num`.

Referenced by `matrix_get_two_norm()`, `moore_penrose_get_pinv()`, `svd_compute_print_U_S_V_s()`, and `trilateration_get_rank_and_homogeneous_solution()`.

### 10.82.2.7 svd\_get\_V\_dim()

```
void svd_get_V_dim (
    uint8_t m,
    uint8_t n,
    matrix_dim_t * v_dim )
```

Calculate the dimension of the matrix V.

#### Parameters

in	<i>m</i>	row number of the matrix to decompose.
in	<i>n</i>	column number of the matrix to decompose.
out	<i>v_dim</i>	pointer to the dimension struct.

Definition at line 108 of file svd.c.

References `matrix_dim_t::col_num`, and `matrix_dim_t::row_num`.

## 10.83 svd.h File Reference

Algorithm for the Singular Value Decomposition (SVD).

```
#include "matrix.h"
```

### Macros

- `#define SVD_COMPUTE_NEGLIGIBLE_VALUES 1`  
*The case of computing negligible values.*
- `#define SVD_SPLIT_AT_NEGLIGIBLE_VALUES 2`  
*The case of splitting at negligible values.*
- `#define SVD_QR_STEP 3`  
*The case of the QR-step.*
- `#define SVD_ORDER_ABSOLUTE_SING_VALUES 4`  
*The case of the order of absolute singular values.*

## Functions

- void `svd` (uint8\_t m, uint8\_t n, `matrix_t` A[m][n], uint8\_t u\_m, uint8\_t u\_n, `matrix_t` U[u\_m][u\_n], `matrix_t` S[u\_n][n], `matrix_t` V[n][n], uint8\_t sing\_vec\_length, `matrix_t` singl\_values\_vec[])  
*Compute the Singular-Value Decomposition (SVD) of a matrix.*
- void `svd_get_U_dim` (uint8\_t m, uint8\_t n, `matrix_dim_t` \*u\_dim)  
*Calculate the dimension of the matrix U.*
- void `svd_get_S_dim` (uint8\_t m, uint8\_t n, `matrix_dim_t` \*s\_dim)  
*Calculate the dimension of the matrix S.*
- void `svd_get_V_dim` (uint8\_t m, uint8\_t n, `matrix_dim_t` \*v\_dim)  
*Calculate the dimension of the matrix V.*
- uint8\_t `svd_get_single_values_num` (uint8\_t m, uint8\_t n)  
*Calculate the number of the singular values.*
- void `svd_get_reciproc_singular_values` (uint8\_t m, uint8\_t n, uint8\_t length, `matrix_t` singl\_values\_arr[], `matrix_t` recip\_singl\_values\_arr[])  
*Compute the reciprocal singular values.*
- void `svd_compute_print_U_S_V_s` (uint8\_t m, uint8\_t n, `matrix_t` matrix\_arr[m][n], uint8\_t i)  
*Compute and print the SVD of a matrix.*

### 10.83.1 Detailed Description

Algorithm for the Singular Value Decomposition (SVD).

Provide necessary methods to compute the compact SVD of a matrix.  $A = U \cdot S \cdot V$ , where U is a (m x l) orthogonal matrix, S is a (l x l) diagonal matrix, V is a (l x n) orthogonal matrix, and  $l = \min(m, n)$ . The SVD is computed by using the Golub–Kahan–Reinsch algorithm that works in two phases: bidiagonalization and a reduction to the diagonal form phase.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.83.2 Function Documentation

#### 10.83.2.1 `svd()`

```
void svd (
    uint8_t m,
    uint8_t n,
    matrix_t A[m][n],
    uint8_t u_m,
    uint8_t u_n,
    matrix_t U[u_m][u_n],
    matrix_t S[u_n][n],
    matrix_t V[n][n],
    uint8_t sing_vec_length,
    matrix_t singl_values_vec[] )
```

Compute the Singular-Value Decomposition (SVD) of a matrix.

Matrix A is transformed to  $A = U \cdot S \cdot V$ , where U and V are unitary matrices, and S is a diagonal matrix.

## Parameters

in	<i>m</i>	row number of the matrix A.
in	<i>n</i>	column number of the matrix A.
in, out	<i>A</i> [][]	pointer to the matrix A.
in	<i>u_m</i>	row number of the matrix U.
in	<i>u_n</i>	column number of the matrix U.
in, out	<i>U</i> [][]	pointer to the matrix U.
in, out	<i>S</i> [][]	pointer to the matrix S.
in, out	<i>V</i> [][]	pointer to the matrix V.
in	<i>sing_vec_length</i>	length of the singular vector.
in, out	<i>singl_values_vec</i> []	pointer to the vector saving the singular values.

Definition at line 119 of file svd.c.

Referenced by `matrix_get_two_norm()`, `svd_compute_print_U_S_V_s()`, and `trilateration_get_rank_and_homogeneous_solution()`.

### 10.83.2.2 svd\_compute\_print\_U\_S\_V\_s()

```
void svd_compute_print_U_S_V_s (
    uint8_t m,
    uint8_t n,
    matrix_t matrix_arr[m][n],
    uint8_t i )
```

Compute and print the SVD of a matrix.

**Parameters**

in	<i>m</i>	row number of the matrix to transform in SVD.
in	<i>n</i>	column number of the matrix to transform in SVD.
in	<i>matrix_arr</i> [][]	pointer to the matrix.
in	<i>i</i>	label.

Definition at line 766 of file svd.c.

References `matrix_dim_t::col_num`, `matrix_print()`, `matrix_t`, `matrix_dim_t::row_num`, `svd()`, `svd_get_single_values_num()`, and `svd_get_U_dim()`.

Referenced by `svd_test()`.

**10.83.2.3 svd\_get\_reciproc\_singular\_values()**

```
void svd_get_reciproc_singular_values (
    uint8_t m,
    uint8_t n,
    uint8_t length,
    matrix_t singl_values_arr[],
    matrix_t recip_singl_values_arr[] )
```

Compute the reciprocal singular values.

This method is based on the singular values.

**Parameters**

in	<i>m</i>	row number of the matrix to transform in SVD.
in	<i>n</i>	column number of the matrix to transform in SVD.
in	<i>length</i>	length of the array of single values.
in	<i>singl_values_arr</i>	pointer to the array of single values.
out	<i>recip_singl_values_arr</i>	pointer to the array of the reciprocal singular values.

Definition at line 747 of file svd.c.

References `matrix_t`.

**10.83.2.4 svd\_get\_S\_dim()**

```
void svd_get_S_dim (
    uint8_t m,
    uint8_t n,
    matrix_dim_t * s_dim )
```

Calculate the dimension of the matrix S.

## Parameters

in	<i>m</i>	row number of the matrix to decompose.
in	<i>n</i>	column number of the matrix to decompose.
out	<i>s_dim</i>	pointer to the dimension struct.

Definition at line 102 of file svd.c.

References `matrix_dim_t::row_num`.

### 10.83.2.5 svd\_get\_single\_values\_num()

```
uint8_t svd_get_single_values_num (
    uint8_t m,
    uint8_t n )
```

Calculate the number of the singular values.

## Parameters

in	<i>m</i>	row number of the matrix to decompose.
in	<i>n</i>	column number of the matrix to decompose.

## Returns

the number of the singular values.

Definition at line 114 of file svd.c.

Referenced by `matrix_get_two_norm()`, `moore_penrose_get_pinv()`, `svd_compute_print_U_S_V_s()`, and `trilateration_get_rank_and_homogeneous_solution()`.

### 10.83.2.6 svd\_get\_U\_dim()

```
void svd_get_U_dim (
    uint8_t m,
    uint8_t n,
    matrix_dim_t * u_dim )
```

Calculate the dimension of the matrix U.

## Parameters

in	<i>m</i>	row number of the matrix to decompose.
in	<i>n</i>	column number of the matrix to decompose.
out	<i>u_dim</i>	pointer to the dimension struct.

Definition at line 96 of file svd.c.

References `matrix_dim_t::col_num`, and `matrix_dim_t::row_num`.

Referenced by `matrix_get_two_norm()`, `moore_penrose_get_pinv()`, `svd_compute_print_U_S_V_s()`, and `trilateration_get_rank_and_homogeneous_solution()`.

### 10.83.2.7 svd\_get\_V\_dim()

```
void svd_get_V_dim (
    uint8_t m,
    uint8_t n,
    matrix_dim_t * v_dim )
```

Calculate the dimension of the matrix V.

#### Parameters

in	<i>m</i>	row number of the matrix to decompose.
in	<i>n</i>	column number of the matrix to decompose.
out	<i>v_dim</i>	pointer to the dimension struct.

Definition at line 108 of file svd.c.

References `matrix_dim_t::col_num`, and `matrix_dim_t::row_num`.

## 10.84 svd\_test.c File Reference

Examples of the SVD algorithm.

```
#include <stdio.h>
#include <inttypes.h>
#include "svd.h"
#include "matrix.h"
#include "vector.h"
```

### Functions

- void [svd\\_test](#) (void)  
*Examples of the Givens algorithm.*

#### 10.84.1 Detailed Description

Examples of the SVD algorithm.

SVD algorithm examples (see the [Singular Value Decomposition \(SVD\)](#) approach).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.85 svd\_test.h File Reference

Examples of the SVD algorithm.

### Functions

- void [svd\\_test](#) (void)  
*Examples of the Givens algorithm.*

### 10.85.1 Detailed Description

Examples of the SVD algorithm.

SVD algorithm examples (see the [Singular Value Decomposition \(SVD\)](#) approach).

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.86 trilateration.c File Reference

Implement the trilateration algorithm.

```
#include <math.h>
#include <stdio.h>
#include <string.h>
#include "moore_penrose_pseudo_inverse.h"
#include "matrix.h"
#include "svd.h"
#include "trilateration.h"
```

### Functions

- void [trilateration1](#) (uint8\_t anchor\_num, [matrix\\_t](#) anchor\_pos\_matrix[anchor\_num][3], [matrix\\_t](#) pseudo\_inv\_matrix[4][anchor\_num], [matrix\\_t](#) homog\_sol\_arr[], [matrix\\_t](#) dist\_arr[], [matrix\\_t](#) solution\_x1[], [matrix\\_t](#) solution\_x2[])  
*Implement the trilateration algorithm using the pre-processed pseudo-inverse matrix.*
- void [trilateration2](#) (uint8\_t anchor\_num, [matrix\\_t](#) anchor\_pos\_matrix[anchor\_num][3], [matrix\\_t](#) dist\_arr[], [matrix\\_t](#) solution\_x1[], [matrix\\_t](#) solution\_x2[])  
*Implement the trilateration algorithm.*
- void [trilateration\\_preprocessed\\_get\\_particular\\_solution](#) ([matrix\\_t](#) pseudo\_inv\_matrix[4][3], [matrix\\_t](#) b\_arr[], [matrix\\_t](#) particular\_solu\_arr[])  
*Compute the particular solution, which is the general solution of  $A\vec{x}_0 = \vec{b}_0$ .*
- uint8\_t [trilateration\\_get\\_rank\\_and\\_homogeneous\\_solution](#) (uint8\_t anchor\_num, [matrix\\_t](#) anchor\_pos\_matrix[anchor\_num][3], [matrix\\_t](#) Xh[])  
*Compute the rank and the solution of the homogeneous system  $A\vec{x}_0 = 0$ .*
- void [trilateration\\_get\\_particular\\_solution](#) (uint8\_t m, uint8\_t n, [matrix\\_t](#) anchor\_pos\_mat[m][n], [matrix\\_t](#) dist\_arr[], [matrix\\_t](#) Xp[])

Compute the particular solution, which is the general solution of  $A\vec{x}_0 = \vec{b}_0$ .

- void `trilateration_get_quadratic_equation_solution` (`matrix_t` Xp[], `matrix_t` Xh[], `matrix_t` solution\_x1[], `matrix_t` solution\_x2[])

Solve a quadratic equation.

- void `trilateration_get_A_matrix` (`uint8_t` m, `matrix_t` anchor\_pos\_matrix[m][3], `matrix_t` A\_matrix[m][4])

Computes the matrix  $A$  of the equation system:  $A\vec{x} = \vec{b}$ .

- void `trilateration_get_b_vector` (`uint8_t` anchor\_num, `matrix_t` dist\_arr[], `matrix_t` anchor\_pos\_↵matrix[anchor\_num][3], `matrix_t` b\_vec[])

Computes the vector  $\vec{b}$  of the equation system:  $A\vec{x} = \vec{b}$ .

- void `trilateration_solve_linear_equation` (`uint8_t` line\_num, `uint8_t` col\_num, `matrix_t` pseudo\_inv\_↵matrix[line\_num][col\_num], `matrix_t` b\_vec[], `matrix_t` sol\_vec[])

Solve a linear equation.

## 10.86.1 Detailed Description

Implement the trilateration algorithm.

### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)  
 Abdelmoumen Norrdine [a.norrdine@googlemail.com](mailto:a.norrdine@googlemail.com)

## 10.86.2 Function Documentation

### 10.86.2.1 trilateration1()

```
void trilateration1 (
    uint8_t anchor_num,
    matrix_t anchor_pos_matrix[anchor_num][3],
    matrix_t pseudo_inv_matrix[4][anchor_num],
    matrix_t homog_sol_arr[],
    matrix_t dist_arr[],
    matrix_t solution_x1[],
    matrix_t solution_x2[] )
```

Implement the trilateration algorithm using the pre-processed pseudo-inverse matrix.

### Note

This function should be initialized with the pre-processed pseudo-inverse matrix of the equation system↵:  $A\vec{x} = 0$ . Caution!: The `solution_x1` and the `solution_x2` vectors have a length of 3 or 4 in the case of two-dimensional or three-dimensional space. The first element ( $x_0$ ) of the `solution_x1[]` and the `solution_x2[]` vectors is a measure of the solvability of the multilateration problem. For example, in the three-dimensional space:  $d = x_0 - (x_1^2 + x_2^2 + x_3^2)$ .

### Parameters

in	<code>anchor_num</code>	number of the reference stations.
----	-------------------------	-----------------------------------

## Parameters

in	<i>anchor_pos_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>pseudo_inv_matrix</i> [][]	pointer to the pre-processed pseudo-inverse matrix.
in	<i>homog_sol_arr</i> []	the homogeneous solution.
in	<i>dist_arr</i> []	distances to the reference stations.
in, out	<i>solution_x1</i> []	includes the first solution. It has the length of 3 or 4 in the case of two-dimensional or three-dimensional space.
in, out	<i>solution_x2</i> []	includes the second solution. It has length of 3 or 4 in the case of two-dimensional or three-dimensional space.

Definition at line 32 of file trilateration.c.

References `matrix_t`, `trilateration_get_b_vector()`, `trilateration_get_quadratic_equation_solution()`, `trilateration_preprocessed_get_particular_solution()`, and `trilateration_solve_linear_equation()`.

Referenced by `magnetic_based_preprocessing_get_position()`.

## 10.86.2.2 trilateration2()

```
void trilateration2 (
    uint8_t anchor_num,
    matrix_t anchor_pos_matrix[anchor_num][3],
    matrix_t dist_arr[],
    matrix_t solution_x1[],
    matrix_t solution_x2[] )
```

Implement the trilateration algorithm.

## Note

The pre-processed pseudo-inverse matrix of the equation system:  $A\vec{x} = 0$  is computed on the mobile station. Caution!: The `solution_x1` and the `solution_x2` vectors have a length of 3 or 4 in the case of two-dimensional or three-dimensional space. The first element ( $x_0$ ) of the `solution_x1`[] and the `solution_x2`[] vectors is a measure of the solvability of the multilateration problem. For example, in the three-dimensional space:  $d = x_0 - (x_1^2 + x_2^2 + x_3^2)$ .

## Parameters

in	<i>anchor_num</i>	number of the reference stations.
in	<i>anchor_pos_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>dist_arr</i> []	distances to the reference stations.
in, out	<i>solution_x1</i> []	includes the first solution. It has the length of 3 or 4 in the case of two-dimensional or three-dimensional space.
in, out	<i>solution_x2</i> []	includes the second solution. It has the length of 3 or 4 in the case of two-dimensional or three-dimensional space.

Definition at line 69 of file trilateration.c.

References `matrix_t`, `moore_penrose_get_pinv()`, `trilateration_get_A_matrix()`, `trilateration_get_b_vector()`, `trilateration_get_particular_solution()`, `trilateration_get_quadratic_equation_solution()`, `trilateration_get_rank_and_homogeneous_solution()`, and `trilateration_solve_linear_equation()`.

Referenced by `distance_based_test()`, `magnetic_based_test()`, and `recog_mitigate_multipath()`.

### 10.86.2.3 trilateration\_get\_A\_matrix()

```
void trilateration_get_A_matrix (
    uint8_t m,
    matrix_t anchor_pos_matrix[m][3],
    matrix_t A_matrix[m][4] )
```

Computes the matrix  $A$  of the equation system:  $A\vec{x} = \vec{b}$ .

#### Parameters

in	<i>m</i>	number of the reference stations.
in	<i>anchor_pos_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in, out	<i>A_matrix</i> [][]	pointer to the matrix $A$ .

Definition at line 256 of file `trilateration.c`.

Referenced by `pos_algos_common_test()`, `trilateration2()`, `trilateration_get_particular_solution()`, and `trilateration_get_rank_and_homogeneous_solution()`.

### 10.86.2.4 trilateration\_get\_b\_vector()

```
void trilateration_get_b_vector (
    uint8_t anchor_num,
    matrix_t dist_arr[],
    matrix_t anchor_pos_matrix[anchor_num][3],
    matrix_t b_vec[] )
```

Computes the vector  $\vec{b}$  of the equation system:  $A\vec{x} = \vec{b}$ .

#### Parameters

in	<i>anchor_num</i>	number of the reference stations.
in	<i>dist_arr</i> []	distances to the reference stations.
in	<i>anchor_pos_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in, out	<i>b_vec</i> []	pointer to the vector <i>vecb</i> .

Definition at line 275 of file `trilateration.c`.

Referenced by `trilateration1()`, `trilateration2()`, and `trilateration_get_particular_solution()`.

### 10.86.2.5 trilateration\_get\_particular\_solution()

```
void trilateration_get_particular_solution (
    uint8_t m,
    uint8_t n,
    matrix_t anchor_pos_mat[m][n],
    matrix_t dist_arr[],
    matrix_t Xp[] )
```

Compute the particular solution, which is the general solution of  $A\vec{x}_0 = \vec{b}_0$ .

#### Note

The particular solution is calculated in the case of three reference stations. The particular solution is computed on the mobile station.

#### Parameters

in	<i>m</i>	number of the reference stations.
in	<i>n</i>	column number of the reference stations matrix.
in	<i>anchor_pos_mat</i> [][]	three-dimensional coordinates of the reference stations.
in	<i>dist_arr</i> []	distances to the reference stations.
in, out	<i>Xp</i> []	includes the particular solution.

Definition at line 179 of file trilateration.c.

References [matrix\\_mul\\_vec\(\)](#), [matrix\\_t](#), [moore\\_penrose\\_get\\_pinv\(\)](#), [trilateration\\_get\\_A\\_matrix\(\)](#), and [trilateration\\_get\\_b\\_vector\(\)](#).

Referenced by [trilateration2\(\)](#).

### 10.86.2.6 trilateration\_get\_quadratic\_equation\_solution()

```
void trilateration_get_quadratic_equation_solution (
    matrix_t particular_solu_arr[],
    matrix_t homogeneous_solution_arr[],
    matrix_t solution_x1[],
    matrix_t solution_x2[] )
```

Solve a quadratic equation.

The quotients of the quadratic equation are derived from the particular and homogeneous solution.

#### Parameters

in	<i>particular_solu_arr</i> []	pointer to the particular solution.
in	<i>homogeneous_solution_arr</i> []	pointer to the homogeneous solution.
in, out	<i>solution_x1</i> []	presents the first solution.
in, out	<i>solution_x2</i> []	presents the second solution.

Definition at line 193 of file trilateration.c.

References `matrix_t`.

Referenced by `trilateration1()`, and `trilateration2()`.

### 10.86.2.7 trilateration\_get\_rank\_and\_homogeneous\_solution()

```
uint8_t trilateration_get_rank_and_homogeneous_solution (
    uint8_t anchor_num,
    matrix_t anchor_pos_matrix[anchor_num][3],
    matrix_t Xh[ ] )
```

Compute the rank and the solution of the homogeneous system  $A\vec{x}_0 = 0$ .

#### Parameters

in	<i>anchor_num</i>	number of the reference stations.
in	<i>anchor_pos_matrix</i> [][]	three-dimensional coordinates of the reference stations.
in, out	<i>Xh</i> []	includes the homogeneous solution.

#### Returns

the rank of the matrix  $A$ .

Definition at line 149 of file trilateration.c.

References `matrix_dim_t::col_num`, `matrix_get_rank()`, `matrix_t`, `matrix_dim_t::row_num`, `svd()`, `svd_get_single_↵`  
`values_num()`, `svd_get_U_dim()`, and `trilateration_get_A_matrix()`.

Referenced by `trilateration2()`.

### 10.86.2.8 trilateration\_preprocessed\_get\_particular\_solution()

```
void trilateration_preprocessed_get_particular_solution (
    matrix_t pseudo_inv_matrix[4][3],
    matrix_t b_arr[],
    matrix_t particular_solu_arr[ ] )
```

Compute the particular solution, which is the general solution of  $A\vec{x}_0 = \vec{b}_0$ .

#### Note

The particular solution is calculated in the case of three reference stations. The particular solution is computed using the pre-processed pseudo-inverse matrix.

## Parameters

in	<i>pseudo_inv_matrix</i> [][]	pointer to the pre-processed pseudo-inverse matrix.
in	<i>b_arr</i> []	pointer the vector <i>vecb</i> .
in, out	<i>particular_solu_arr</i> []	includes the particular solution.

Definition at line 121 of file trilateration.c.

References `matrix_t`.

Referenced by `trilateration1()`.

### 10.86.2.9 trilateration\_solve\_linear\_equation()

```
void trilateration_solve_linear_equation (
    uint8_t line_num,
    uint8_t col_num,
    matrix_t pseudo_inv_matrix[line_num][col_num],
    matrix_t b_vec[],
    matrix_t sol_vec[] )
```

Solve a linear equation.

The linear equation is solved by using the pre-processed pseudo-inverse matrix and the vector  $\vec{b}$

## Parameters

in	<i>line_num</i>	row number of the pseudo-inverse matrix.
in	<i>col_num</i>	column number of the pseudo-inverse matrix.
in	<i>pseudo_inv_matrix</i> [][]	pointer to the pre-processed pseudo-inverse matrix.
in	<i>b_vec</i> []	pointer to the vector $\vec{b}$ .
out	<i>sol_vec</i> []	solution vector.

Definition at line 291 of file trilateration.c.

References `matrix_t`.

Referenced by `trilateration1()`, and `trilateration2()`.

## 10.87 trilateration.h File Reference

Implement the trilateration algorithm.

```
#include <inttypes.h>
#include "matrix.h"
```

## Functions

- void `trilateration1` (uint8\_t anchor\_num, `matrix_t` anchor\_pos\_matrix[anchor\_num][3], `matrix_t` pseudo\_inv\_matrix[4][anchor\_num], `matrix_t` homog\_sol\_arr[], `matrix_t` dist\_arr[], `matrix_t` solution\_x1[], `matrix_t` solution\_x2[])

*Implement the trilateration algorithm using the pre-processed pseudo-inverse matrix.*

- void `trilateration2` (uint8\_t anchor\_num, `matrix_t` anchor\_pos\_matrix[anchor\_num][3], `matrix_t` dist\_arr[], `matrix_t` solution\_x1[], `matrix_t` solution\_x2[])

*Implement the trilateration algorithm.*

- void `trilateration_get_A_matrix` (uint8\_t m, `matrix_t` anchor\_pos\_matrix[m][3], `matrix_t` A\_matrix[m][4])

*Computes the matrix  $A$  of the equation system:  $A\vec{x} = \vec{b}$ .*

- void `trilateration_get_b_vector` (uint8\_t anchor\_num, `matrix_t` dist\_arr[], `matrix_t` anchor\_pos\_matrix[anchor\_num][3], `matrix_t` b\_vec[])

*Computes the vector  $\vec{b}$  of the equation system:  $A\vec{x} = \vec{b}$ .*

- void `trilateration_preprocessed_get_particular_solution` (`matrix_t` pseudo\_inv\_matrix[4][3], `matrix_t` b\_arr[], `matrix_t` particular\_solu\_arr[])

*Compute the particular solution, which is the general solution of  $A\vec{x}_0 = \vec{b}_0$ .*

- void `trilateration_get_particular_solution` (uint8\_t m, uint8\_t n, `matrix_t` anchor\_pos\_mat[m][n], `matrix_t` dist\_arr[], `matrix_t` Xp[])

*Compute the particular solution, which is the general solution of  $A\vec{x}_0 = \vec{b}_0$ .*

- uint8\_t `trilateration_get_rank_and_homogeneous_solution` (uint8\_t anchor\_num, `matrix_t` anchor\_pos\_matrix[anchor\_num][3], `matrix_t` Xh[])

*Compute the rank and the solution of the homogeneous system  $A\vec{x}_0 = 0$ .*

- void `trilateration_get_quadratic_equation_solution` (`matrix_t` particular\_solu\_arr[], `matrix_t` homogeneous\_solution\_arr[], `matrix_t` solution\_x1[], `matrix_t` solution\_x2[])

*Solve a quadratic equation.*

- void `trilateration_solve_linear_equation` (uint8\_t line\_num, uint8\_t col\_num, `matrix_t` pseudo\_inv\_matrix[line\_num][col\_num], `matrix_t` b\_vec[], `matrix_t` sol\_vec[])

*Solve a linear equation.*

### 10.87.1 Detailed Description

Implement the trilateration algorithm.

#### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

Abdelmoumen Norrdine [a.norrdine@gmail.com](mailto:a.norrdine@gmail.com)

### 10.87.2 Function Documentation

### 10.87.2.1 trilateration1()

```
void trilateration1 (
    uint8_t anchor_num,
    matrix_t anchor_pos_matrix[anchor_num][3],
    matrix_t pseudo_inv_matrix[4][anchor_num],
    matrix_t homog_sol_arr[],
    matrix_t dist_arr[],
    matrix_t solution_x1[],
    matrix_t solution_x2[] )
```

Implement the trilateration algorithm using the pre-processed pseudo-inverse matrix.

#### Note

This function should be initialized with the pre-processed pseudo-inverse matrix of the equation system  $A\vec{x} = 0$ . Caution!: The `solution_x1` and the `solution_x2` vectors have a length of 3 or 4 in the case of two-dimensional or three-dimensional space. The first element ( $x_0$ ) of the `solution_x1[]` and the `solution_x2[]` vectors is a measure of the solvability of the multilateration problem. For example, in the three-dimensional space:  $d = x_0 - (x_1^2 + x_2^2 + x_3^2)$ .

#### Parameters

in	<code>anchor_num</code>	number of the reference stations.
in	<code>anchor_pos_matrix[][]</code>	three-dimensional coordinates of the reference stations.
in	<code>pseudo_inv_matrix[][]</code>	pointer to the pre-processed pseudo-inverse matrix.
in	<code>homog_sol_arr[]</code>	the homogeneous solution.
in	<code>dist_arr[]</code>	distances to the reference stations.
in, out	<code>solution_x1[]</code>	includes the first solution. It has the length of 3 or 4 in the case of two-dimensional or three-dimensional space.
in, out	<code>solution_x2[]</code>	includes the second solution. It has length of 3 or 4 in the case of two-dimensional or three-dimensional space.

Definition at line 32 of file `trilateration.c`.

References `matrix_t`, `trilateration_get_b_vector()`, `trilateration_get_quadratic_equation_solution()`, `trilateration_preprocessed_get_particular_solution()`, and `trilateration_solve_linear_equation()`.

Referenced by `magnetic_based_preprocessing_get_position()`.

### 10.87.2.2 trilateration2()

```
void trilateration2 (
    uint8_t anchor_num,
    matrix_t anchor_pos_matrix[anchor_num][3],
    matrix_t dist_arr[],
    matrix_t solution_x1[],
    matrix_t solution_x2[] )
```

Implement the trilateration algorithm.

**Note**

The pre-processed pseudo-inverse matrix of the equation system:  $A\vec{x} = 0$  is computed on the mobile station. Caution!: The `solution_x1` and the `solution_x2` vectors have a length of 3 or 4 in the case of two-dimensional or three-dimensional space. The first element ( $x_0$ ) of the `solution_x1[]` and the `solution_x2[]` vectors is a measure of the solvability of the multilateration problem. For example, in the three-dimensional space:  $d = x_0 - (x_1^2 + x_2^2 + x_3^2)$ .

**Parameters**

in	<code>anchor_num</code>	number of the reference stations.
in	<code>anchor_pos_matrix[][]</code>	three-dimensional coordinates of the reference stations.
in	<code>dist_arr[]</code>	distances to the reference stations.
in, out	<code>solution_x1[]</code>	includes the first solution. It has the length of 3 or 4 in the case of two-dimensional or three-dimensional space.
in, out	<code>solution_x2[]</code>	includes the second solution. It has the length of 3 or 4 in the case of two-dimensional or three-dimensional space.

Definition at line 69 of file `trilateration.c`.

References `matrix_t`, `moore_penrose_get_pinv()`, `trilateration_get_A_matrix()`, `trilateration_get_b_vector()`, `trilateration_get_particular_solution()`, `trilateration_get_quadratic_equation_solution()`, `trilateration_get_rank_and_homogeneous_solution()`, and `trilateration_solve_linear_equation()`.

Referenced by `distance_based_test()`, `magnetic_based_test()`, and `recog_mitigate_multipath()`.

**10.87.2.3 trilateration\_get\_A\_matrix()**

```
void trilateration_get_A_matrix (
    uint8_t m,
    matrix_t anchor_pos_matrix[m][3],
    matrix_t A_matrix[m][4] )
```

Computes the matrix  $A$  of the equation system:  $A\vec{x} = \vec{b}$ .

**Parameters**

in	<code>m</code>	number of the reference stations.
in	<code>anchor_pos_matrix[][]</code>	three-dimensional coordinates of the reference stations.
in, out	<code>A_matrix[][]</code>	pointer to the matrix $A$ .

Definition at line 256 of file `trilateration.c`.

Referenced by `pos_algos_common_test()`, `trilateration2()`, `trilateration_get_particular_solution()`, and `trilateration_get_rank_and_homogeneous_solution()`.

### 10.87.2.4 trilateration\_get\_b\_vector()

```
void trilateration_get_b_vector (
    uint8_t anchor_num,
    matrix_t dist_arr[],
    matrix_t anchor_pos_matrix[anchor_num][3],
    matrix_t b_vec[] )
```

Computes the vector  $\vec{b}$  of the equation system:  $A\vec{x} = \vec{b}$ .

#### Parameters

in	<i>anchor_num</i>	number of the reference stations.
in	<i>dist_arr[]</i>	distances to the reference stations.
in	<i>anchor_pos_matrix[][]</i>	three-dimensional coordinates of the reference stations.
in, out	<i>b_vec[]</i>	pointer to the vector <i>vecb</i> .

Definition at line 275 of file trilateration.c.

Referenced by `trilateration1()`, `trilateration2()`, and `trilateration_get_particular_solution()`.

### 10.87.2.5 trilateration\_get\_particular\_solution()

```
void trilateration_get_particular_solution (
    uint8_t m,
    uint8_t n,
    matrix_t anchor_pos_mat[m][n],
    matrix_t dist_arr[],
    matrix_t Xp[] )
```

Compute the particular solution, which is the general solution of  $A\vec{x}_0 = \vec{b}_0$ .

#### Note

The particular solution is calculated in the case of three reference stations. The particular solution is computed on the mobile station.

#### Parameters

in	<i>m</i>	number of the reference stations.
in	<i>n</i>	column number of the reference stations matrix.
in	<i>anchor_pos_mat[][]</i>	three-dimensional coordinates of the reference stations.
in	<i>dist_arr[]</i>	distances to the reference stations.
in, out	<i>Xp[]</i>	includes the particular solution.

Definition at line 179 of file trilateration.c.

References `matrix_mul_vec()`, `matrix_t`, `moore_penrose_get_pinv()`, `trilateration_get_A_matrix()`, and `trilateration_get_b_vector()`.

Referenced by `trilateration2()`.

### 10.87.2.6 `trilateration_get_quadratic_equation_solution()`

```
void trilateration_get_quadratic_equation_solution (
    matrix_t particular_solu_arr[],
    matrix_t homogeneous_solution_arr[],
    matrix_t solution_x1[],
    matrix_t solution_x2[] )
```

Solve a quadratic equation.

The quotients of the quadratic equation are derived from the particular and homogeneous solution.

#### Parameters

in	<i>particular_solu_arr[]</i>	pointer to the particular solution.
in	<i>homogeneous_solution_arr[]</i>	pointer to the homogeneous solution.
in, out	<i>solution_x1[]</i>	presents the first solution.
in, out	<i>solution_x2[]</i>	presents the second solution.

Definition at line 193 of file `trilateration.c`.

References `matrix_t`.

Referenced by `trilateration1()`, and `trilateration2()`.

### 10.87.2.7 `trilateration_get_rank_and_homogeneous_solution()`

```
uint8_t trilateration_get_rank_and_homogeneous_solution (
    uint8_t anchor_num,
    matrix_t anchor_pos_matrix[anchor_num][3],
    matrix_t Xh[] )
```

Compute the rank and the solution of the homogeneous system  $A\vec{x}_0 = 0$ .

#### Parameters

in	<i>anchor_num</i>	number of the reference stations.
in	<i>anchor_pos_matrix[][]</i>	three-dimensional coordinates of the reference stations.
in, out	<i>Xh[]</i>	includes the homogeneous solution.

#### Returns

the rank of the matrix  $A$ .

Definition at line 149 of file `trilateration.c`.

References `matrix_dim_t::col_num`, `matrix_get_rank()`, `matrix_t`, `matrix_dim_t::row_num`, `svd()`, `svd_get_single_↵`  
`values_num()`, `svd_get_U_dim()`, and `trilateration_get_A_matrix()`.

Referenced by `trilateration2()`.

### 10.87.2.8 trilateration\_preprocessed\_get\_particular\_solution()

```
void trilateration_preprocessed_get_particular_solution (
    matrix_t pseudo_inv_matrix[4][3],
    matrix_t b_arr[],
    matrix_t particular_solu_arr[] )
```

Compute the particular solution, which is the general solution of  $A\vec{x}_0 = \vec{b}_0$ .

#### Note

The particular solution is calculated in the case of three reference stations. The particular solution is computed using the pre-processed pseudo-inverse matrix.

#### Parameters

in	<code>pseudo_inv_matrix[][]</code>	pointer to the pre-processed pseudo-inverse matrix.
in	<code>b_arr[]</code>	pointer the vector <i>vecb</i> .
in, out	<code>particular_solu_arr[]</code>	includes the particular solution.

Definition at line 121 of file `trilateration.c`.

References `matrix_t`.

Referenced by `trilateration1()`.

### 10.87.2.9 trilateration\_solve\_linear\_equation()

```
void trilateration_solve_linear_equation (
    uint8_t line_num,
    uint8_t col_num,
    matrix_t pseudo_inv_matrix[line_num][col_num],
    matrix_t b_vec[],
    matrix_t sol_vec[] )
```

Solve a linear equation.

The linear equation is solved by using the pre-processed pseudo-inverse matrix and the vector  $\vec{b}$

## Parameters

in	<i>line_num</i>	row number of the pseudo-inverse matrix.
in	<i>col_num</i>	column number of the pseudo-inverse matrix.
in	<i>pseudo_inv_matrix[ ][ ]</i>	pointer to the pre-processed pseudo-inverse matrix.
in	<i>b_vec[ ]</i>	pointer to the vector $\vec{b}$ .
out	<i>sol_vec[ ]</i>	solution vector.

Definition at line 291 of file trilateration.c.

References `matrix_t`.

Referenced by `trilateration1()`, and `trilateration2()`.

## 10.88 utils.c File Reference

Utilities for linear algebra. Utility-functions are needed by the linear algebra-module as well as other modules such as the position algorithm-module.

```
#include <math.h>
#include <stdarg.h>
#include <stdio.h>
#include "utils.h"
#include "shell_sort.h"
```

### Functions

- double [utils\\_to\\_radian](#) (double deg\_angle)  
*Convert the angle from degrees to radians.*
- double [utils\\_sind](#) (double deg\_angle)  
*Compute the sine of a variable in degrees.*
- void [utils\\_swap](#) (uint8\_t \*a, uint8\_t \*b)  
*Interchange the values of two variables of type uint8\_t.*
- double [utils\\_max](#) (double a, double b)  
*Returns the greater of two real numbers.*
- double [utils\\_min](#) (double a, double b)  
*Returns the smaller of two real numbers.*
- uint8\_t [utils\\_u8\\_max](#) (uint8\_t a, uint8\_t b)  
*Returns the greater of two numbers from type uint8\_t.*
- uint8\_t [utils\\_u8\\_min](#) (uint8\_t a, uint8\_t b)  
*Returns the smaller of two numbers from type uint8\_t.*
- void [utils\\_printf](#) (char \*format\_str,...)  
*Print by using variable format string as well as argument lists.*
- double [utils\\_mean](#) (uint8\_t arr\_size, [vector\\_t](#) in\_arr[ ])  
*Compute the mean value of a data set.*
- void [utils\\_moving\\_average](#) (uint8\_t arr\_size, [vector\\_t](#) in\_arr[ ], uint8\_t window\_size, [vector\\_t](#) out\_arr[ ])  
*Compute the moving average of a data set.*
- double [utils\\_get\\_median](#) ([vector\\_t](#) arr[ ], uint8\_t length)  
*Compute the median of a finite array of numbers.*
- double [utils\\_get\\_save\\_square\\_root](#) (double x, double y)  
 *$\sqrt{a^2 + b^2}$  without under/overflow.*

## 10.88.1 Detailed Description

Utilities for linear algebra. Utility-functions are needed by the linear algebra-module as well as other modules such as the position algorithm-module.

### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.88.2 Function Documentation

### 10.88.2.1 `utils_get_median()`

```
double utils_get_median (
    vector_t arr[],
    uint8_t length )
```

Compute the median of a finite array of numbers.

#### Parameters

in	<i>arr[]</i>	pointer to the data set.
in	<i>length</i>	size of the data set.

#### Returns

the median value of the data set.

Definition at line 152 of file `utils.c`.

References `shell_sort()`.

Referenced by `utils_test()`.

### 10.88.2.2 `utils_get_save_square_root()`

```
double utils_get_save_square_root (
    double x,
    double y )
```

$\sqrt{a^2 + b^2}$  without under/overflow.

Compute the square root without under/overflow.

Definition at line 168 of file `utils.c`.

Referenced by `qr_householder_decomp()`.

### 10.88.2.3 `utils_max()`

```
double utils_max (
    double a,
    double b )
```

Returns the greater of two real numbers.

#### Parameters

in	<i>a</i>	the first value to compare.
in	<i>b</i>	the second value to compare.

#### Returns

the greater of *a* and *b*.

Definition at line 55 of file `utils.c`.

Referenced by `loc_gauss_newton()`, `modified_gauss_newton()`, and `utils_test()`.

### 10.88.2.4 `utils_mean()`

```
double utils_mean (
    uint8_t arr_size,
    vector_t in_arr[] )
```

Compute the mean value of a data set.

#### Parameters

in	<i>arr_size</i>	size of the data set.
in	<i>in_arr[]</i>	pointer to the data set.

#### Returns

the mean value of the data set.

Definition at line 105 of file `utils.c`.

Referenced by `utils_test()`.

### 10.88.2.5 `utils_min()`

```
double utils_min (
    double a,
    double b )
```

Returns the smaller of two real numbers.

**Parameters**

in	<i>a</i>	the first value to compare.
in	<i>b</i>	the second value to compare.

**Returns**

the smaller of *a* and *b*.

Definition at line 65 of file `utils.c`.

Referenced by `utils_test()`.

**10.88.2.6 utils\_moving\_average()**

```
void utils_moving_average (
    uint8_t arr_size,
    vector_t in_arr[],
    uint8_t window_size,
    vector_t out_arr[] )
```

Compute the moving average of a data set.

**Parameters**

in	<i>arr_size</i>	size of the data set.
in	<i>in_arr[]</i>	pointer to the data set.
in	<i>window_size</i>	window size.
out	<i>out_arr</i>	pointer to the values of the moving average.

Definition at line 121 of file `utils.c`.

Referenced by `utils_test()`.

**10.88.2.7 utils\_printf()**

```
void utils_printf (
    char * format_str,
    ... )
```

Print by using variable format string as well as argument lists.

This function enables to print data by using a variable format string as well as argument list. Furthermore, it avoids the error: "format not a string literal", if `printf` is used.

**Parameters**

in	<i>*format_str</i>	format string.
in	...	argument list.

Definition at line 96 of file utils.c.

Referenced by `matrix_flex_part_print()`, `matrix_flex_print()`, and `vector_flex_print()`.

**10.88.2.8   utils\_sind()**

```
double utils_sind (  
    double deg_angle )
```

Compute the sine of a variable in degrees.

Calculate the sine of the variable `deg_angle`, which is expressed in degrees.

**Parameters**

in	<i>deg_angle</i>	angle in degrees.
----	------------------	-------------------

**Returns**

sine value.

Definition at line 39 of file utils.c.

References `M_PI`.

Referenced by `utils_test()`.

**10.88.2.9   utils\_swap()**

```
void utils_swap (  
    uint8_t * a,  
    uint8_t * b )
```

Interchange the values of two variables of type `uint8_t`.

**Parameters**

in	<i>*a</i>	pointer to first variable.
in	<i>*b</i>	pointer to second variable.

Definition at line 46 of file utils.c.

Referenced by `utils_test()`.

#### 10.88.2.10 `utils_to_radian()`

```
double utils_to_radian (  
    double deg_angle )
```

Convert the angle from degrees to radians.

##### Parameters

in	<i>deg_angle</i>	angle in degrees.
----	------------------	-------------------

##### Returns

angle in radians.

Definition at line 31 of file utils.c.

References `M_PI`.

Referenced by `magnetic_based_test()`, and `utils_test()`.

#### 10.88.2.11 `utils_u8_max()`

```
uint8_t utils_u8_max (  
    uint8_t a,  
    uint8_t b )
```

Returns the greater of two numbers from type `uint8_t`.

##### Parameters

in	<i>a</i>	the first value to compare.
in	<i>b</i>	the second value to compare.

##### Returns

the greater of *a* and *b*.

Definition at line 75 of file utils.c.

### 10.88.2.12 `utils_u8_min()`

```
uint8_t utils_u8_min (
    uint8_t a,
    uint8_t b )
```

Returns the smaller of two numbers from type `uint8_t`.

#### Parameters

in	<i>a</i>	the first value to compare.
in	<i>b</i>	the second value to compare.

#### Returns

the smaller of *a* and *b*.

Definition at line 85 of file `utils.c`.

## 10.89 `utils.h` File Reference

Utilities for linear algebra.

```
#include <inttypes.h>
#include "vector.h"
```

### Macros

- `#define M_PI 3.14159265358979323846`  
*Define the pi-constant.*

### Functions

- double `utils_to_radian` (double *deg\_angle*)  
*Convert the angle from degrees to radians.*
- double `utils_sind` (double *deg\_angle*)  
*Compute the sine of a variable in degrees.*
- void `utils_swap` (uint8\_t \**a*, uint8\_t \**b*)  
*Interchange the values of two variables of type uint8\_t.*
- double `utils_max` (double *a*, double *b*)  
*Returns the greater of two real numbers.*
- double `utils_min` (double *a*, double *b*)  
*Returns the smaller of two real numbers.*
- uint8\_t `utils_u8_max` (uint8\_t *a*, uint8\_t *b*)  
*Returns the greater of two numbers from type uint8\_t.*
- uint8\_t `utils_u8_min` (uint8\_t *a*, uint8\_t *b*)  
*Returns the smaller of two numbers from type uint8\_t.*

- void `utils_printf` (char \*format\_str,...)  
*Print by using variable format string as well as argument lists.*
- double `utils_mean` (uint8\_t arr\_size, `vector_t` in\_arr[])  
*Compute the mean value of a data set.*
- void `utils_moving_average` (uint8\_t arr\_size, `vector_t` in\_arr[], uint8\_t window\_size, `vector_t` out\_arr[])  
*Compute the moving average of a data set.*
- double `utils_get_median` (`vector_t` arr[], uint8\_t length)  
*Compute the median of a finite array of numbers.*
- double `utils_get_save_square_root` (double x, double y)  
*Compute the square root without under/overflow.*

## 10.89.1 Detailed Description

Utilities for linear algebra.

Utility-functions are needed by linear algebra-module as well as other modules such as the position algorithm-module.

### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.89.2 Function Documentation

### 10.89.2.1 `utils_get_median()`

```
double utils_get_median (  
    vector_t arr[],  
    uint8_t length )
```

Compute the median of a finite array of numbers.

#### Parameters

in	<code>arr[]</code>	pointer to the data set.
in	<code>length</code>	size of the data set.

#### Returns

the median value of the data set.

Definition at line 152 of file `utils.c`.

References `shell_sort()`.

Referenced by `utils_test()`.

### 10.89.2.2 `utils_get_save_square_root()`

```
double utils_get_save_square_root (
    double x,
    double y )
```

Compute the square root without under/overflow.

#### Parameters

in	<i>x</i>	first value.
in	<i>y</i>	second value.

#### Returns

save square root.

Compute the square root without under/overflow.

Definition at line 168 of file `utils.c`.

Referenced by `qr_householder_decomp()`.

### 10.89.2.3 `utils_max()`

```
double utils_max (
    double a,
    double b )
```

Returns the greater of two real numbers.

#### Parameters

in	<i>a</i>	the first value to compare.
in	<i>b</i>	the second value to compare.

#### Returns

the greater of *a* and *b*.

Definition at line 55 of file `utils.c`.

Referenced by `loc_gauss_newton()`, `modified_gauss_newton()`, and `utils_test()`.

#### 10.89.2.4 utils\_mean()

```
double utils_mean (
    uint8_t arr_size,
    vector_t in_arr[] )
```

Compute the mean value of a data set.

##### Parameters

in	<i>arr_size</i>	size of the data set.
in	<i>in_arr[]</i>	pointer to the data set.

##### Returns

the mean value of the data set.

Definition at line 105 of file utils.c.

Referenced by `utils_test()`.

#### 10.89.2.5 utils\_min()

```
double utils_min (
    double a,
    double b )
```

Returns the smaller of two real numbers.

##### Parameters

in	<i>a</i>	the first value to compare.
in	<i>b</i>	the second value to compare.

##### Returns

the smaller of a and b.

Definition at line 65 of file utils.c.

Referenced by `utils_test()`.

#### 10.89.2.6 utils\_moving\_average()

```
void utils_moving_average (
    uint8_t arr_size,
```

```
vector_t in_arr[],
uint8_t window_size,
vector_t out_arr[] )
```

Compute the moving average of a data set.

#### Parameters

in	<i>arr_size</i>	size of the data set.
in	<i>in_arr[]</i>	pointer to the data set.
in	<i>window_size</i>	window size.
out	<i>out_arr</i>	pointer to the values of the moving average.

Definition at line 121 of file `utils.c`.

Referenced by `utils_test()`.

#### 10.89.2.7 `utils_printf()`

```
void utils_printf (
    char * format_str,
    ... )
```

Print by using variable format string as well as argument lists.

This function enables to print data by using a variable format string as well as argument list. Furthermore, it avoids the error: "format not a string literal", if `printf` is used.

#### Parameters

in	<i>*format_str</i>	format string.
in	...	argument list.

Definition at line 96 of file `utils.c`.

Referenced by `matrix_flex_part_print()`, `matrix_flex_print()`, and `vector_flex_print()`.

#### 10.89.2.8 `utils_sind()`

```
double utils_sind (
    double deg_angle )
```

Compute the sine of a variable in degrees.

Calculate the sine of the variable `deg_angle`, which is expressed in degrees.

**Parameters**

in	<i>deg_angle</i>	angle in degrees.
----	------------------	-------------------

**Returns**

sine value.

Definition at line 39 of file utils.c.

References M\_PI.

Referenced by utils\_test().

**10.89.2.9   utils\_swap()**

```
void utils_swap (
    uint8_t * a,
    uint8_t * b )
```

Interchange the values of two variables of type uint8\_t.

**Parameters**

in	<i>*a</i>	pointer to first variable.
in	<i>*b</i>	pointer to second variable.

Definition at line 46 of file utils.c.

Referenced by utils\_test().

**10.89.2.10   utils\_to\_radian()**

```
double utils_to_radian (
    double deg_angle )
```

Convert the angle from degrees to radians.

**Parameters**

in	<i>deg_angle</i>	angle in degrees.
----	------------------	-------------------

**Returns**

angle in radians.

Definition at line 31 of file utils.c.

References `M_PI`.

Referenced by `magnetic_based_test()`, and `utils_test()`.

#### 10.89.2.11 `utils_u8_max()`

```
uint8_t utils_u8_max (  
    uint8_t a,  
    uint8_t b )
```

Returns the greater of two numbers from type `uint8_t`.

##### Parameters

in	<i>a</i>	the first value to compare.
in	<i>b</i>	the second value to compare.

##### Returns

the greater of *a* and *b*.

Definition at line 75 of file utils.c.

#### 10.89.2.12 `utils_u8_min()`

```
uint8_t utils_u8_min (  
    uint8_t a,  
    uint8_t b )
```

Returns the smaller of two numbers from type `uint8_t`.

##### Parameters

in	<i>a</i>	the first value to compare.
in	<i>b</i>	the second value to compare.

##### Returns

the smaller of *a* and *b*.

Definition at line 85 of file utils.c.

## 10.90 `utils_test.c` File Reference

Examples of the utility functions.

```
#include <stdio.h>
#include "utils.h"
#include "vector.h"
```

### Functions

- void `utils_test` (void)  
*Examples of the utility functions.*

#### 10.90.1 Detailed Description

Examples of the utility functions.

Utility examples (see [utilities](#) functions).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.91 `utils_test.h` File Reference

Examples of the utility functions.

### Functions

- void `utils_test` (void)  
*Examples of the utility functions.*

#### 10.91.1 Detailed Description

Examples of the utility functions.

Utility examples (see [utilities](#) functions).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.92 vector.c File Reference

Vector computations. Vector computations include operations such as addition, subtraction, and inner product (dot product).

```
#include <inttypes.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include "vector.h"
#include "utils.h"
```

### Functions

- void [vector\\_clear](#) (uint8\_t n, [vector\\_t](#) arr[])  
*Clear all the elements of the vector.*
- void [vector\\_copy](#) (uint8\_t size, [vector\\_t](#) src\_arr[], [vector\\_t](#) dest\_arr[])  
*Copy the elements of the source vector to the destination vector.*
- [vector\\_t](#) [vector\\_get\\_norm2](#) (uint8\_t length, [vector\\_t](#) arr[])  
*Compute the 2-norm norm of a vector.*
- [vector\\_t](#) [vector\\_get\\_square\\_norm2](#) (uint8\_t length, [vector\\_t](#) arr[])  
*Compute the squared 2-norm norm of a vector .*
- [vector\\_t](#) [vector\\_get\\_sum](#) (uint8\_t length, [vector\\_t](#) arr[])  
*Compute the sum of the elements of a vector.*
- [vector\\_t](#) [vector\\_get\\_mean\\_value](#) (uint8\_t length, [vector\\_t](#) arr[])  
*Compute the average or mean value of a vector.*
- void [vector\\_sub](#) (uint8\_t size, [vector\\_t](#) a\_vec[], [vector\\_t](#) b\_vec[], [vector\\_t](#) a\_minus\_b[])  
*Compute the subtraction of two vectors.*
- void [vector\\_add](#) (uint8\_t size, [vector\\_t](#) a\_vec[size], [vector\\_t](#) b\_vec[size], [vector\\_t](#) a\_plus\_b\_vec[size])  
*Compute the addition of two vectors.*
- void [vector\\_mul](#) (uint8\_t size, [vector\\_t](#) a\_vec[size], [vector\\_t](#) b\_vec[size], [vector\\_t](#) a\_mul\_b\_vec[size])  
*Compute the multiplication of two vectors.*
- void [vector\\_square](#) (uint8\_t n, [vector\\_t](#) vec[n], [vector\\_t](#) square\_vec[n])  
*Compute the square of a vector.*
- void [vector\\_in\\_place\\_scalar\\_mul](#) (uint8\_t size, [vector\\_t](#) a\_vec[size], [vector\\_t](#) scl)  
*Compute the product of a vector with a real number.*
- void [vector\\_scalar\\_mul](#) (uint8\_t size, [vector\\_t](#) src\_vec[size], [vector\\_t](#) scl, [vector\\_t](#) dest\_vec[])  
*Compute the product of a vector with a real number.*
- void [vector\\_scalar\\_div](#) (uint8\_t size, [vector\\_t](#) a\_vec[size], [vector\\_t](#) scl)  
*Compute the division of a vector with a real number.*
- [vector\\_t](#) [vector\\_get\\_euclidean\\_distance](#) (uint8\_t length, [vector\\_t](#) vec1[], [vector\\_t](#) vec2[])  
*Compute the Euclidean distance between two vectors.*
- [vector\\_t](#) [vector\\_get\\_max\\_and\\_index](#) (uint8\_t length, [vector\\_t](#) vec[], uint8\_t \*index)  
*Compute the maximal value and its index of a vector.*
- [vector\\_t](#) [vector\\_get\\_scalar\\_product](#) (uint8\_t n, [vector\\_t](#) vec1[n], [vector\\_t](#) vec2[n])  
*Compute the dot product of two vectors.*
- bool [vector\\_is\\_equal](#) (uint16\_t length, [vector\\_t](#) vec\_1[], [vector\\_t](#) vec\_2[])  
*Determine the equality of two vectors.*

- void `vector_get_index_vector` (uint8\_t k, uint8\_t n, `vector_t` unsorted\_vector[n], `vector_t` sorted\_vector[n], uint8\_t index\_vector[n])  
*Determine the index of the vector elements before sorting.*
- `vector_t` `vector_get_residual` (uint8\_t length, `vector_t` a\_vec[], `vector_t` b\_vec[])  
*Compute the residual of two vectors.*
- void `vector_get_elements` (`vector_t` src\_vec[], uint8\_t k, uint8\_t index\_vec[], `vector_t` dst\_vec[])  
*Get the elements of the vector by an index vector.*
- bool `vector_uint32_is_equal` (uint32\_t length, uint32\_t vec\_1[], uint32\_t vec\_2[])  
*Determine the equality of two vectors of type uint32\_t.*
- void `vector_print` (uint32\_t length, `vector_t` arr[])  
*Display the values of the vector's elements.*
- void `vector_print_u8_array` (uint32\_t length, uint8\_t arr[])  
*Display the values of the vector's elements of type uint8\_t.*
- void `vector_flex_print` (uint32\_t length, `vector_t` arr[], uint8\_t before\_dot, uint8\_t after\_dot)  
*Display the values of the vector's elements.*

## 10.92.1 Detailed Description

Vector computations. Vector computations include operations such as addition, subtraction, and inner product (dot product).

### Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.92.2 Function Documentation

### 10.92.2.1 `vector_add()`

```
void vector_add (
    uint8_t size,
    vector_t a_vec[size],
    vector_t b_vec[size],
    vector_t a_plus_b_vec[size] )
```

Compute the addition of two vectors.

Add b\_vec to a\_vec and return the result in a\_plus\_b.

#### Parameters

in	size	number of elements to subtract.
in	a_vec[]	pointer to the first vector.
in	b_vec[]	pointer to the second vector.
out	a_plus_b_vec[]	pointer to the destination vector.

Definition at line 104 of file vector.c.

Referenced by `damped_newton_raphson()`, `loc_levenberg_marquardt()`, `loc_levenberg_marquardt_correction()`, `opt_levenberg_marquardt()`, `opt_levenberg_marquardt_correction()`, and `vector_test()`.

### 10.92.2.2 `vector_clear()`

```
void vector_clear (
    uint8_t size,
    vector_t arr[] )
```

Clear all the elements of the vector.

#### Parameters

in	<i>size</i>	size of the vector.
in	<i>arr[]</i>	pointer to the vector.

Definition at line 32 of file vector.c.

References `vector_t`.

Referenced by `dist_based_jacobian_get_JTf()`, `fsolve_test()`, `optimization_exponential_data_test()`, `optimization_↔sinusoidal_data_test()`, `optimization_test()`, `position_optimization_test()`, and `vector_test()`.

### 10.92.2.3 `vector_copy()`

```
void vector_copy (
    uint8_t size,
    vector_t src_arr[],
    vector_t dest_arr[] )
```

Copy the elements of the source vector to the destination vector.

#### Parameters

in	<i>size</i>	number of elements to copy.
in	<i>src_arr[]</i>	pointer to the source vector.
in	<i>dest_arr[]</i>	pointer to the destination vector.

Definition at line 37 of file vector.c.

References `vector_t`.

Referenced by `damped_newton_raphson()`, `loc_gauss_newton()`, `loc_levenberg_marquardt()`, `modified_gauss_↔newton()`, `multipath_algo_own_norm_distr_test()`, `newton_raphson()`, `opt_levenberg_marquardt()`, `recog_mitigate_↔_multipath()`, and `vector_test()`.

### 10.92.2.4 vector\_flex\_print()

```
void vector_flex_print (
    uint32_t length,
    vector_t arr[],
    uint8_t before_dot,
    uint8_t after_dot )
```

Display the values of the vector's elements.

This function allows the user to determine the precision as well as the with of the numbers to display.

#### Parameters

in	<i>length</i>	size of the vector to display.
in	<i>arr</i>	pointer to the vector.
in	<i>before_dot</i>	the number of digits to be printed before the decimal point.
in	<i>after_dot</i>	the number of digits to be printed after the decimal point.

Definition at line 284 of file vector.c.

References `utils_printf()`.

Referenced by `distance_based_test()`, `fsolve_test()`, `magnetic_based_get_magnetic_field()`, `magnetic_based_test()`, `multipath_algo_own_norm_distr_test()`, `optimization_test()`, `position_optimization_test()`, `solve_big_matrix_test()`, `solve_test()`, `utils_test()`, and `vector_test()`.

### 10.92.2.5 vector\_get\_elements()

```
void vector_get_elements (
    vector_t src_vec[],
    uint8_t k,
    uint8_t index_vec[],
    vector_t dst_vec[] )
```

Get the elements of the vector by an index vector.

#### Parameters

in	<i>src_vec[]</i>	pointer to source vector.
in	<i>k</i>	size of the index vector.
in	<i>index_vec[]</i>	pointer to the index vector.
out	<i>dst_vec[]</i>	pointer to the destination vector

Definition at line 235 of file vector.c.

Referenced by `multipath_algo_own_norm_distr_test()`.

### 10.92.2.6 vector\_get\_euclidean\_distance()

```
vector_t vector_get_euclidean_distance (
    uint8_t length,
    vector_t vec1[],
    vector_t vec2[] )
```

Compute the Euclidean distance between two vectors.

#### Parameters

in	<i>length</i>	size of the vector.
in	<i>vec1[]</i>	pointer to the first vector.
in	<i>vec2[]</i>	pointer to the second vector.

#### Returns

the Euclidean distance.

Definition at line 163 of file vector.c.

References `vector_get_norm2()`, `vector_sub()`, and `vector_t`.

Referenced by `loc_gauss_newton()`, `modified_gauss_newton()`, `newton_raphson()`, and `vector_test()`.

### 10.92.2.7 vector\_get\_index\_vector()

```
void vector_get_index_vector (
    uint8_t k,
    uint8_t n,
    vector_t unsorted_vector[n],
    vector_t sorted_vector[n],
    uint8_t index_vector[n] )
```

Determine the index of the vector elements before sorting.

Determine the index of the elements of a sorted vector. These indices correspond to the positions of the elements in the unsorted vector.

#### Parameters

in	<i>k</i>	size of the unsorted vector.
in	<i>n</i>	size of the sorted vector.
in	<i>unsorted_vector[]</i>	pointer to the unsorted vector.
in	<i>sorted_vector[]</i>	pointer to the sorted vector.
out	<i>index_vector[]</i>	pointer to the index vector.

Definition at line 214 of file vector.c.

Referenced by `recog_mitigate_multipath()`.

### 10.92.2.8 vector\_get\_max\_and\_index()

```
vector_t vector_get_max_and_index (
    uint8_t length,
    vector_t vec[],
    uint8_t * index )
```

Compute the maximal value and its index of a vector.

#### Parameters

in	<i>length</i>	vector size.
in	<i>vec[]</i>	pointer to the vector.
in	<i>index</i>	pointer to the index.

#### Returns

the maximal value of the vector.

Definition at line 175 of file vector.c.

References `vector_t`.

### 10.92.2.9 vector\_get\_mean\_value()

```
vector_t vector_get_mean_value (
    uint8_t length,
    vector_t arr[] )
```

Compute the average or mean value of a vector.

#### Parameters

in	<i>length</i>	size of the vector.
in	<i>arr[]</i>	pointer to the vector.

#### Returns

the mean value of the vector.

Definition at line 78 of file vector.c.

References `vector_t`.

Referenced by `vector_test()`.

**10.92.2.10 vector\_get\_norm2()**

```
vector_t vector_get_norm2 (
    uint8_t length,
    vector_t arr[] )
```

Compute the 2-norm norm of a vector.

**Parameters**

in	<i>length</i>	size of the vector.
in	<i>arr[]</i>	pointer to the vector.

**Returns**

the 2-norm of the vector.

Definition at line 42 of file vector.c.

References vector\_t.

Referenced by damped\_newton\_raphson(), dist\_based\_get\_distance\_to\_anchor(), get\_damped\_norm(), loc\_↔  
gauss\_newton(), loc\_levenberg\_marquardt(), magnetic\_based\_get\_distances\_to\_anchors(), modified\_gauss\_↔  
newton(), opt\_levenberg\_marquardt(), recog\_mitigate\_multipath(), vector\_get\_euclidean\_distance(), vector\_get\_↔  
\_residual(), and vector\_test().

**10.92.2.11 vector\_get\_residual()**

```
vector_t vector_get_residual (
    uint8_t length,
    vector_t a_vec[],
    vector_t b_vec[] )
```

Compute the residual of two vectors.

**Parameters**

in	<i>length</i>	vector size.
in	<i>a_vec[]</i>	pointer to the first vector.
in	<i>b_vec[]</i>	pointer to the second vector.

**Returns**

the residual of the two vectors.

Definition at line 227 of file vector.c.

References vector\_get\_norm2(), vector\_sub(), and vector\_t.

### 10.92.2.12 vector\_get\_scalar\_product()

```
vector_t vector_get_scalar_product (
    uint8_t n,
    vector_t vec1[n],
    vector_t vec2[n] )
```

Compute the dot product of two vectors.

#### Parameters

in	<i>n</i>	size of the vectors.
in	<i>vec1[]</i>	pointer to the first vector.
in	<i>vec2[]</i>	pointer to the second vector.

#### Returns

the scalar product of two vectors.

Definition at line 190 of file vector.c.

References `vector_t`.

Referenced by `loc_levenberg_marquardt_correction()`, `opt_levenberg_marquardt_correction()`, and `vector_test()`.

### 10.92.2.13 vector\_get\_square\_norm2()

```
vector_t vector_get_square_norm2 (
    uint8_t length,
    vector_t arr[] )
```

Compute the squared 2-norm norm of a vector .

#### Parameters

in	<i>length</i>	size of the vector.
in	<i>arr[]</i>	pointer to the vector.

#### Returns

the squared 2-norm of the vector.

Definition at line 54 of file vector.c.

References `vector_t`.

Referenced by `vector_test()`.

**10.92.2.14 vector\_get\_sum()**

```
vector_t vector_get_sum (
    uint8_t length,
    vector_t arr[] )
```

Compute the sum of the elements of a vector.

**Parameters**

in	<i>length</i>	size of the vector.
in	<i>arr[]</i>	pointer to the vector.

**Returns**

the sum of the elements of the vector.

Definition at line 66 of file vector.c.

References `vector_t`.

Referenced by `vector_test()`.

**10.92.2.15 vector\_in\_place\_scalar\_mul()**

```
void vector_in_place_scalar_mul (
    uint8_t size,
    vector_t a_vec[size],
    vector_t scl )
```

Compute the product of a vector with a real number.

Multiple the elements of a vector with a scalar and return the result in the vector itself.

**Parameters**

in	<i>size</i>	number of elements to multiply with a scalar.
in, out	<i>a_vec[]</i>	pointer to the source/destination vector.
in	<i>scl</i>	a scalar.

Definition at line 131 of file vector.c.

Referenced by `get_delta_x()`.

**10.92.2.16 vector\_is\_equal()**

```
bool vector_is_equal (
    uint16_t length,
```

```
vector_t vec_1[],  
vector_t vec_2[] )
```

Determine the equality of two vectors.

#### Parameters

in	<i>length</i>	size of the vector.
in	<i>vec_1[]</i>	pointer to the first vector.
in	<i>vec_2[]</i>	pointer to the second vector.

#### Returns

true, if the two vectors are equal.

false, if not.

Definition at line 202 of file vector.c.

Referenced by vector\_test().

### 10.92.2.17 vector\_mul()

```
void vector_mul (  
    uint8_t size,  
    vector_t a_vec[size],  
    vector_t b_vec[size],  
    vector_t a_mul_b_vec[size] )
```

Compute the multiplication of two vectors.

Multiple vectors *a\_vec* and *b\_vec* element by element and return the result in *a\_mul\_b*.

#### Parameters

in	<i>size</i>	number of elements to multiply.
in	<i>a_vec[]</i>	pointer to the first vector.
in	<i>b_vec[]</i>	pointer to the second vector.
out	<i>a_mul_b_vec[]</i>	pointer to the destination vector.

Definition at line 114 of file vector.c.

Referenced by vector\_test().

### 10.92.2.18 vector\_print()

```
void vector_print (  
    uint32_t length,  
    vector_t arr[] )
```

Display the values of the vector's elements.

#### Parameters

in	<i>length</i>	size of the vector to display.
in	<i>arr</i>	pointer to the vector.

Definition at line 255 of file vector.c.

#### 10.92.2.19 vector\_print\_u8\_array()

```
void vector_print_u8_array (
    uint32_t length,
    uint8_t arr[] )
```

Display the values of the vector's elements of type uint8\_t.

#### Parameters

in	<i>length</i>	size of the vector to display.
in	<i>arr</i>	pointer to the vector.

Definition at line 269 of file vector.c.

#### 10.92.2.20 vector\_scalar\_div()

```
void vector_scalar_div (
    uint8_t size,
    vector_t a_vec[size],
    vector_t scl )
```

Compute the division of a vector with a real number.

Divide the elements of a vector with a scalar and return the result in the vector itself.

#### Parameters

in	<i>size</i>	number of elements to divide with a scalar.
in, out	<i>a_vec[]</i>	pointer to the source/destination vector.
in	<i>scl</i>	a scalar.

Definition at line 151 of file vector.c.

### 10.92.2.21 vector\_scalar\_mul()

```
void vector_scalar_mul (
    uint8_t size,
    vector_t src_vec[size],
    vector_t scl,
    vector_t dest_vec[] )
```

Compute the product of a vector with a real number.

Multiple the elements of a vector with a scalar and return the result in other vector.

#### Parameters

in	<i>size</i>	number of elements to multiply with a scalar.
in	<i>src_vec[]</i>	pointer to the source vector.
in	<i>scl</i>	a scalar.
out	<i>dest_vec</i>	pointer to the destination vector.

Definition at line 141 of file vector.c.

Referenced by `damped_newton_raphson()`.

### 10.92.2.22 vector\_square()

```
void vector_square (
    uint8_t n,
    vector_t vec[n],
    vector_t square_vec[n] )
```

Compute the square of a vector.

Square the elements of vector `vec` and return the result in `square_vec`.

#### Parameters

in	<i>n</i>	number of elements to square.
in	<i>vec[]</i>	pointer to the source vector.
out	<i>square_vec[]</i>	pointer to the destination vector.

Definition at line 124 of file vector.c.

Referenced by `recog_mitigate_multipath()`.

### 10.92.2.23 vector\_sub()

```
void vector_sub (
    uint8_t size,
```

```
vector_t a_vec[ ],
vector_t b_vec[ ],
vector_t a_minus_b[ ] )
```

Compute the subtraction of two vectors.

Subtract `b_vec` from `a_vec` and return the result in `a_minus_b`.

#### Parameters

in	<i>size</i>	number of elements to subtract.
in	<i>a_vec[]</i>	pointer to the first vector.
in	<i>b_vec[]</i>	pointer to the second vector.
out	<i>a_minus_b[]</i>	pointer to the destination vector.

Definition at line 94 of file `vector.c`.

Referenced by `dist_based_get_distance_to_anchor()`, `loc_gauss_newton()`, `magnetic_based_get_distances_to_anchors()`, `modified_gauss_newton()`, `newton_raphson()`, `recog_mitigate_multipath()`, `vector_get_euclidean_distance()`, `vector_get_residual()`, and `vector_test()`.

#### 10.92.2.24 vector\_uint32\_is\_equal()

```
bool vector_uint32_is_equal (
    uint32_t length,
    uint32_t vec_1[],
    uint32_t vec_2[] )
```

Determine the equality of two vectors of type `uint32_t`.

#### Parameters

in	<i>length</i>	size of the vector.
in	<i>vec_1[]</i>	pointer to the first vector.
in	<i>vec_2[]</i>	pointer to the second vector.

#### Returns

true, if the two vectors are equal.  
false, if not.

Definition at line 243 of file `vector.c`.

## 10.93 vector.h File Reference

Vector computations.

```
#include <inttypes.h>
#include <stdbool.h>
```

## Macros

- #define `vector_t` double  
Define the data type of the vector elements.

## Functions

- void `vector_clear` (uint8\_t size, `vector_t` arr[])  
Clear all the elements of the vector.
- void `vector_copy` (uint8\_t size, `vector_t` src\_arr[], `vector_t` dest\_arr[])  
Copy the elements of the source vector to the destination vector.
- `vector_t` `vector_get_norm2` (uint8\_t length, `vector_t` arr[])  
Compute the 2-norm norm of a vector.
- `vector_t` `vector_get_square_norm2` (uint8\_t length, `vector_t` arr[])  
Compute the squared 2-norm norm of a vector.
- `vector_t` `vector_get_sum` (uint8\_t length, `vector_t` arr[])  
Compute the sum of the elements of a vector.
- `vector_t` `vector_get_mean_value` (uint8\_t length, `vector_t` arr[])  
Compute the average or mean value of a vector.
- void `vector_sub` (uint8\_t size, `vector_t` a\_vec[], `vector_t` b\_vec[], `vector_t` a\_minus\_b[])  
Compute the subtraction of two vectors.
- void `vector_add` (uint8\_t size, `vector_t` a\_vec[size], `vector_t` b\_vec[size], `vector_t` a\_plus\_b\_vec[size])  
Compute the addition of two vectors.
- void `vector_mul` (uint8\_t size, `vector_t` a\_vec[size], `vector_t` b\_vec[size], `vector_t` a\_mul\_b\_vec[size])  
Compute the multiplication of two vectors.
- void `vector_square` (uint8\_t n, `vector_t` vec[n], `vector_t` square\_vec[n])  
Compute the square of a vector.
- void `vector_in_place_scalar_mul` (uint8\_t size, `vector_t` a\_vec[size], `vector_t` scl)  
Compute the product of a vector with a real number.
- void `vector_scalar_mul` (uint8\_t size, `vector_t` src\_vec[size], `vector_t` scl, `vector_t` dest\_vec[])  
Compute the product of a vector with a real number.
- void `vector_scalar_div` (uint8\_t size, `vector_t` a\_vec[size], `vector_t` scl)  
Compute the division of a vector with a real number.
- `vector_t` `vector_get_euclidean_distance` (uint8\_t length, `vector_t` vec1[], `vector_t` vec2[])  
Compute the Euclidean distance between two vectors.
- `vector_t` `vector_get_scalar_product` (uint8\_t n, `vector_t` vec1[n], `vector_t` vec2[n])  
Compute the dot product of two vectors.
- bool `vector_is_equal` (uint16\_t length, `vector_t` vec\_1[], `vector_t` vec\_2[])  
Determine the equality of two vectors.
- bool `vector_uint32_is_equal` (uint32\_t length, uint32\_t vec\_1[], uint32\_t vec\_2[])  
Determine the equality of two vectors of type `uint32_t`.
- void `vector_get_index_vector` (uint8\_t k, uint8\_t n, `vector_t` unsorted\_vector[n], `vector_t` sorted\_vector[n], uint8\_t index\_vector[n])  
Determine the index of the vector elements before sorting.
- `vector_t` `vector_get_max_and_index` (uint8\_t length, `vector_t` vec[], uint8\_t \*index)  
Compute the maximal value and its index of a vector.
- `vector_t` `vector_get_residual` (uint8\_t length, `vector_t` a\_vec[], `vector_t` b\_vec[])  
Compute the residual of two vectors.
- void `vector_get_elements` (`vector_t` src\_vec[], uint8\_t k, uint8\_t index\_vec[], `vector_t` dst\_vec[])  
Get the elements of the vector by an index vector.

- void `vector_print` (uint32\_t length, `vector_t` arr[])  
*Display the values of the vector's elements.*
- void `vector_print_u8_array` (uint32\_t length, uint8\_t arr[])  
*Display the values of the vector's elements of type uint8\_t.*
- void `vector_flex_print` (uint32\_t length, `vector_t` arr[], uint8\_t before\_dot, uint8\_t after\_dot)  
*Display the values of the vector's elements.*

### 10.93.1 Detailed Description

Vector computations.

Vector computations include operations such as addition, subtraction, and inner product (dot product).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

### 10.93.2 Function Documentation

#### 10.93.2.1 `vector_add()`

```
void vector_add (
    uint8_t size,
    vector_t a_vec[size],
    vector_t b_vec[size],
    vector_t a_plus_b_vec[size] )
```

Compute the addition of two vectors.

Add `b_vec` to `a_vec` and return the result in `a_plus_b`.

Parameters

in	<code>size</code>	number of elements to subtract.
in	<code>a_vec[]</code>	pointer to the first vector.
in	<code>b_vec[]</code>	pointer to the second vector.
out	<code>a_plus_b_vec[]</code>	pointer to the destination vector.

Definition at line 104 of file `vector.c`.

Referenced by `damped_newton_raphson()`, `loc_levenberg_marquardt()`, `loc_levenberg_marquardt_correction()`, `opt_levenberg_marquardt()`, `opt_levenberg_marquardt_correction()`, and `vector_test()`.

### 10.93.2.2 vector\_clear()

```
void vector_clear (
    uint8_t size,
    vector_t arr[] )
```

Clear all the elements of the vector.

#### Parameters

in	<i>size</i>	size of the vector.
in	<i>arr[]</i>	pointer to the vector.

Definition at line 32 of file vector.c.

References `vector_t`.

Referenced by `dist_based_jacobian_get_JTf()`, `fsolve_test()`, `optimization_exponential_data_test()`, `optimization_↵ sinusoidal_data_test()`, `optimization_test()`, `position_optimization_test()`, and `vector_test()`.

### 10.93.2.3 vector\_copy()

```
void vector_copy (
    uint8_t size,
    vector_t src_arr[],
    vector_t dest_arr[] )
```

Copy the elements of the source vector to the destination vector.

#### Parameters

in	<i>size</i>	number of elements to copy.
in	<i>src_arr[]</i>	pointer to the source vector.
in	<i>dest_arr[]</i>	pointer to the destination vector.

Definition at line 37 of file vector.c.

References `vector_t`.

Referenced by `damped_newton_raphson()`, `loc_gauss_newton()`, `loc_levenberg_marquardt()`, `modified_gauss_↵ newton()`, `multipath_algo_own_norm_distr_test()`, `newton_raphson()`, `opt_levenberg_marquardt()`, `recog_mitigate_↵ _multipath()`, and `vector_test()`.

### 10.93.2.4 vector\_flex\_print()

```
void vector_flex_print (
    uint32_t length,
```

```
vector_t arr[],
uint8_t before_dot,
uint8_t after_dot )
```

Display the values of the vector's elements.

This function allows the user to determine the precision as well as the with of the numbers to display.

#### Parameters

in	<i>length</i>	size of the vector to display.
in	<i>arr</i>	pointer to the vector.
in	<i>before_dot</i>	the number of digits to be printed before the decimal point.
in	<i>after_dot</i>	the number of digits to be printed after the decimal point.

Definition at line 284 of file vector.c.

References `utils_printf()`.

Referenced by `distance_based_test()`, `fsolve_test()`, `magnetic_based_get_magnetic_field()`, `magnetic_based_test()`, `multipath_algo_own_norm_distr_test()`, `optimization_test()`, `position_optimization_test()`, `solve_big_matrix_test()`, `solve_test()`, `utils_test()`, and `vector_test()`.

#### 10.93.2.5 vector\_get\_elements()

```
void vector_get_elements (
    vector_t src_vec[],
    uint8_t k,
    uint8_t index_vec[],
    vector_t dst_vec[] )
```

Get the elements of the vector by an index vector.

#### Parameters

in	<i>src_vec[]</i>	pointer to source vector.
in	<i>k</i>	size of the index vector.
in	<i>index_vec[]</i>	pointer to the index vector.
out	<i>dst_vec[]</i>	pointer to the destination vector

Definition at line 235 of file vector.c.

Referenced by `multipath_algo_own_norm_distr_test()`.

#### 10.93.2.6 vector\_get\_euclidean\_distance()

```
vector_t vector_get_euclidean_distance (
    uint8_t length,
```

```
vector_t vec1[ ],
vector_t vec2[ ] )
```

Compute the Euclidean distance between two vectors.

#### Parameters

in	<i>length</i>	size of the vector.
in	<i>vec1[]</i>	pointer to the first vector.
in	<i>vec2[]</i>	pointer to the second vector.

#### Returns

the Euclidean distance.

Definition at line 163 of file vector.c.

References `vector_get_norm2()`, `vector_sub()`, and `vector_t`.

Referenced by `loc_gauss_newton()`, `modified_gauss_newton()`, `newton_raphson()`, and `vector_test()`.

#### 10.93.2.7 vector\_get\_index\_vector()

```
void vector_get_index_vector (
    uint8_t k,
    uint8_t n,
    vector_t unsorted_vector[n],
    vector_t sorted_vector[n],
    uint8_t index_vector[n] )
```

Determine the index of the vector elements before sorting.

Determine the index of the elements of a sorted vector. These indices correspond to the positions of the elements in the unsorted vector.

#### Parameters

in	<i>k</i>	size of the unsorted vector.
in	<i>n</i>	size of the sorted vector.
in	<i>unsorted_vector[]</i>	pointer to the unsorted vector.
in	<i>sorted_vector[]</i>	pointer to the sorted vector.
out	<i>index_vector[]</i>	pointer to the index vector.

Definition at line 214 of file vector.c.

Referenced by `recog_mitigate_multipath()`.

### 10.93.2.8 vector\_get\_max\_and\_index()

```
vector_t vector_get_max_and_index (
    uint8_t length,
    vector_t vec[],
    uint8_t * index )
```

Compute the maximal value and its index of a vector.

#### Parameters

in	<i>length</i>	vector size.
in	<i>vec[]</i>	pointer to the vector.
in	<i>index</i>	pointer to the index.

#### Returns

the maximal value of the vector.

Definition at line 175 of file vector.c.

References vector\_t.

### 10.93.2.9 vector\_get\_mean\_value()

```
vector_t vector_get_mean_value (
    uint8_t length,
    vector_t arr[] )
```

Compute the average or mean value of a vector.

#### Parameters

in	<i>length</i>	size of the vector.
in	<i>arr[]</i>	pointer to the vector.

#### Returns

the mean value of the vector.

Definition at line 78 of file vector.c.

References vector\_t.

Referenced by vector\_test().

### 10.93.2.10 vector\_get\_norm2()

```
vector_t vector_get_norm2 (
    uint8_t length,
    vector_t arr[] )
```

Compute the 2-norm norm of a vector.

#### Parameters

in	<i>length</i>	size of the vector.
in	<i>arr[]</i>	pointer to the vector.

#### Returns

the 2-norm of the vector.

Definition at line 42 of file vector.c.

References `vector_t`.

Referenced by `damped_newton_raphson()`, `dist_based_get_distance_to_anchor()`, `get_damped_norm()`, `loc_↔  
gauss_newton()`, `loc_levenberg_marquardt()`, `magnetic_based_get_distances_to_anchors()`, `modified_gauss_↔  
newton()`, `opt_levenberg_marquardt()`, `recog_mitigate_multipath()`, `vector_get_euclidean_distance()`, `vector_get_↔  
_residual()`, and `vector_test()`.

### 10.93.2.11 vector\_get\_residual()

```
vector_t vector_get_residual (
    uint8_t length,
    vector_t a_vec[],
    vector_t b_vec[] )
```

Compute the residual of two vectors.

#### Parameters

in	<i>length</i>	vector size.
in	<i>a_vec[]</i>	pointer to the first vector.
in	<i>b_vec[]</i>	pointer to the second vector.

#### Returns

the residual of the two vectors.

Definition at line 227 of file vector.c.

References `vector_get_norm2()`, `vector_sub()`, and `vector_t`.

**10.93.2.12 vector\_get\_scalar\_product()**

```
vector_t vector_get_scalar_product (
    uint8_t n,
    vector_t vec1[n],
    vector_t vec2[n] )
```

Compute the dot product of two vectors.

**Parameters**

in	<i>n</i>	size of the vectors.
in	<i>vec1[]</i>	pointer to the first vector.
in	<i>vec2[]</i>	pointer to the second vector.

**Returns**

the scalar product of two vectors.

Definition at line 190 of file vector.c.

References `vector_t`.

Referenced by `loc_levenberg_marquardt_correction()`, `opt_levenberg_marquardt_correction()`, and `vector_test()`.

**10.93.2.13 vector\_get\_square\_norm2()**

```
vector_t vector_get_square_norm2 (
    uint8_t length,
    vector_t arr[] )
```

Compute the squared 2-norm norm of a vector .

**Parameters**

in	<i>length</i>	size of the vector.
in	<i>arr[]</i>	pointer to the vector.

**Returns**

the squared 2-norm of the vector.

Definition at line 54 of file vector.c.

References `vector_t`.

Referenced by `vector_test()`.

**10.93.2.14 vector\_get\_sum()**

```
vector_t vector_get_sum (
    uint8_t length,
    vector_t arr[] )
```

Compute the sum of the elements of a vector.

**Parameters**

in	<i>length</i>	size of the vector.
in	<i>arr[]</i>	pointer to the vector.

**Returns**

the sum of the elements of the vector.

Definition at line 66 of file vector.c.

References `vector_t`.

Referenced by `vector_test()`.

**10.93.2.15 vector\_in\_place\_scalar\_mul()**

```
void vector_in_place_scalar_mul (
    uint8_t size,
    vector_t a_vec[size],
    vector_t scl )
```

Compute the product of a vector with a real number.

Multiple the elements of a vector with a scalar and return the result in the vector itself.

**Parameters**

in	<i>size</i>	number of elements to multiply with a scalar.
in, out	<i>a_vec[]</i>	pointer to the source/destination vector.
in	<i>scl</i>	a scalar.

Definition at line 131 of file vector.c.

Referenced by `get_delta_x()`.

**10.93.2.16 vector\_is\_equal()**

```
bool vector_is_equal (
    uint16_t length,
```

```
vector_t vec_1[ ],
vector_t vec_2[ ] )
```

Determine the equality of two vectors.

#### Parameters

in	<i>length</i>	size of the vector.
in	<i>vec_1[]</i>	pointer to the first vector.
in	<i>vec_2[]</i>	pointer to the second vector.

#### Returns

true, if the two vectors are equal.

false, if not.

Definition at line 202 of file vector.c.

Referenced by vector\_test().

### 10.93.2.17 vector\_mul()

```
void vector_mul (
    uint8_t size,
    vector_t a_vec[size],
    vector_t b_vec[size],
    vector_t a_mul_b_vec[size] )
```

Compute the multiplication of two vectors.

Multiple vectors *a\_vec* and *b\_vec* element by element and return the result in *a\_mul\_b*.

#### Parameters

in	<i>size</i>	number of elements to multiply.
in	<i>a_vec[]</i>	pointer to the first vector.
in	<i>b_vec[]</i>	pointer to the second vector.
out	<i>a_mul_b_vec[]</i>	pointer to the destination vector.

Definition at line 114 of file vector.c.

Referenced by vector\_test().

### 10.93.2.18 vector\_print()

```
void vector_print (
    uint32_t length,
    vector_t arr[ ] )
```

Display the values of the vector's elements.

**Parameters**

in	<i>length</i>	size of the vector to display.
in	<i>arr</i>	pointer to the vector.

Definition at line 255 of file vector.c.

**10.93.2.19 vector\_print\_u8\_array()**

```
void vector_print_u8_array (
    uint32_t length,
    uint8_t arr[] )
```

Display the values of the vector's elements of type uint8\_t.

**Parameters**

in	<i>length</i>	size of the vector to display.
in	<i>arr</i>	pointer to the vector.

Definition at line 269 of file vector.c.

**10.93.2.20 vector\_scalar\_div()**

```
void vector_scalar_div (
    uint8_t size,
    vector_t a_vec[size],
    vector_t scl )
```

Compute the division of a vector with a real number.

Divide the elements of a vector with a scalar and return the result in the vector itself.

**Parameters**

in	<i>size</i>	number of elements to divide with a scalar.
in, out	<i>a_vec[]</i>	pointer to the source/destination vector.
in	<i>scl</i>	a scalar.

Definition at line 151 of file vector.c.

**10.93.2.21 vector\_scalar\_mul()**

```
void vector_scalar_mul (
    uint8_t size,
    vector_t src_vec[size],
    vector_t scl,
    vector_t dest_vec[] )
```

Compute the product of a vector with a real number.

Multiple the elements of a vector with a scalar and return the result in other vector.

**Parameters**

in	<i>size</i>	number of elements to multiply with a scalar.
in	<i>src_vec[]</i>	pointer to the source vector.
in	<i>scl</i>	a scalar.
out	<i>dest_vec</i>	pointer to the destination vector.

Definition at line 141 of file vector.c.

Referenced by `damped_newton_raphson()`.

**10.93.2.22 vector\_square()**

```
void vector_square (
    uint8_t n,
    vector_t vec[n],
    vector_t square_vec[n] )
```

Compute the square of a vector.

Square the elements of vector `vec` and return the result in `square_vec`.

**Parameters**

in	<i>n</i>	number of elements to square.
in	<i>vec[]</i>	pointer to the source vector.
out	<i>square_vec[]</i>	pointer to the destination vector.

Definition at line 124 of file vector.c.

Referenced by `recog_mitigate_multipath()`.

**10.93.2.23 vector\_sub()**

```
void vector_sub (
    uint8_t size,
```

```
vector_t a_vec[ ],
vector_t b_vec[ ],
vector_t a_minus_b[ ] )
```

Compute the subtraction of two vectors.

Subtract `b_vec` from `a_vec` and return the result in `a_minus_b`.

#### Parameters

in	<i>size</i>	number of elements to subtract.
in	<i>a_vec[]</i>	pointer to the first vector.
in	<i>b_vec[]</i>	pointer to the second vector.
out	<i>a_minus_b[]</i>	pointer to the destination vector.

Definition at line 94 of file `vector.c`.

Referenced by `dist_based_get_distance_to_anchor()`, `loc_gauss_newton()`, `magnetic_based_get_distances_to_anchors()`, `modified_gauss_newton()`, `newton_raphson()`, `recog_mitigate_multipath()`, `vector_get_euclidean_distance()`, `vector_get_residual()`, and `vector_test()`.

#### 10.93.2.24 vector\_uint32\_is\_equal()

```
bool vector_uint32_is_equal (
    uint32_t length,
    uint32_t vec_1[],
    uint32_t vec_2[] )
```

Determine the equality of two vectors of type `uint32_t`.

#### Parameters

in	<i>length</i>	size of the vector.
in	<i>vec_1[]</i>	pointer to the first vector.
in	<i>vec_2[]</i>	pointer to the second vector.

#### Returns

true, if the two vectors are equal.  
false, if not.

Definition at line 243 of file `vector.c`.

## 10.94 vector\_test.c File Reference

Examples of vector computations.

```
#include <stdio.h>
#include <stdint.h>
#include <inttypes.h>
#include <math.h>
#include "vector.h"
```

## Functions

- void [vector\\_test](#) (void)  
*Examples of vector operations.*

### 10.94.1 Detailed Description

Examples of vector computations.

Vector computation examples of the [vector.h](#) functions.

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)

## 10.95 [vector\\_test.h](#) File Reference

Examples of vector computations.

## Functions

- void [vector\\_test](#) (void)  
*Examples of vector operations.*

### 10.95.1 Detailed Description

Examples of vector computations.

Vector computation examples (see [vector](#) functions).

Author

Zakaria Kasmi [zkasmi@inf.fu-berlin.de](mailto:zkasmi@inf.fu-berlin.de)