QuadOpt

v1.0

Generated by Doxygen 1.8.9.1

Tue May 26 2015 16:54:50

Contents

1 Data Structure Index 1

1.1 Data Structures 1

2 File Index 3

2.1 File List 3

3 Data Structure Documentation 5

3.1 matrix Struct Reference 5

3.1.1 Detailed Description 5

3.1.2 Field Documentation 5

3.1.2.1 columns 5

3.1.2.2 diagonals 5

3.1.2.3 rows 5

3.1.2.4 size 5

3.1.2.5 start 5

3.2 problem Struct Reference 5

3.2.1 Detailed Description 6

3.2.2 Field Documentation 6

3.2.2.1 A 6

3.2.2.2 accuracy 6

3.2.2.3 active_set 6

3.2.2.4 b 7

3.2.2.5 check_time 7

3.2.2.6 constraints_count 7

3.2.2.7 E 7

3.2.2.8 equality_count 7

3.2.2.9 F 7

3.2.2.10 g 7

3.2.2.11 gk 7

3.2.2.12 h 7

3.2.2.13 has_solution 7

3.2.2.14 has_start_point 7

3.2.2.15 inequality_count 7
3.2.2.16 is_sparse 8
3.2.2.17 lagrange 8
3.2.2.18 max_iter 8
3.2.2.19 max_micro_sec 8
3.2.2.20 p 8
3.2.2.21 Q 8
3.2.2.22 q 8
3.2.2.23 Q_inv 8
3.2.2.24 solution 8
3.2.2.25 solution_value 8
3.2.2.26 sparse_A 8
3.2.2.27 sparse_E 8
3.2.2.28 sparse_F 8
3.2.2.29 sparse_Q 8
3.2.2.30 sparse_Q_inv 8
3.2.2.31 step 8
3.2.2.32 variable_count 8
3.2.2.33 z 9
3.2.2.34 z0 9
3.3 sparse_matrix Struct Reference 9
3.3.1 Detailed Description 9
3.3.2 Field Documentation 9
3.3.2.1 A 9
3.3.2.2 cA 9
3.3.2.3 columns 9
3.3.2.4 rA 9
3.3.2.5 rows 9
3.3.2.6 size 9
3.4 work_set Struct Reference 9
3.4.1 Detailed Description 10
3.4.2 Field Documentation 10
3.4.2.1 count 10
3.4.2.2 data 10
3.4.2.3 max_count 10

4 File Documentation 11

4.1 TDDD77/matlab/quadopt.c File Reference 11
4.1.1 Function Documentation 11
4.1.1.1 mexFunction 11

4.2 TDDD77/matrixlibrary/include/matLib.h File Reference 11

4.2.1 Macro Definition Documentation 13

4.2.1.1 DOUBLE 13

4.2.1.2 FORMAT_STRING 13

4.2.1.3 PRECISION 13

4.2.2 Typedef Documentation 13

4.2.2.1 matrix 13

4.2.2.2 value 13

4.2.3 Function Documentation 13

4.2.3.1 add_matrices 13

4.2.3.2 add_matrices_with_return 14

4.2.3.3 add_rows 14

4.2.3.4 check_boundaries 14

4.2.3.5 compare_elements 14

4.2.3.6 compare_matrices 14

4.2.3.7 create_identity_matrix 14

4.2.3.8 create_matrix 14

4.2.3.9 create_zero_matrix 14

4.2.3.10 crout 14

4.2.3.11 derivate_matrix_with_return 14

4.2.3.12 divide_column_with_scalar 14

4.2.3.13 divide_matrix_with_scalar 14

4.2.3.14 divide_row_with_scalar 15

4.2.3.15 dot_product 15

4.2.3.16 first_nonezero_in_column_index 15

4.2.3.17 first_nonezero_in_row_index 15

4.2.3.18 forward_backward 15

4.2.3.19 free_matrix 15

4.2.3.20 gauss_jordan 15

4.2.3.21 gauss_jordan_solver 15

4.2.3.22 get_column_vector 15

4.2.3.23 get_column_vector_with_return 15

4.2.3.24 get_determinant 15

4.2.3.25 get_diagonal 15

4.2.3.26 get_inverse 16

4.2.3.27 get_inverse_of_2x2 16

4.2.3.28 get_inverse_of_2x2_with_return 16

4.2.3.29 get_matrix_with_only_pivots 16

4.2.3.30 get_row_vector 16

4.2.3.31 get_row_vector_with_return 16

4.2.3.32 get_sub_matrix 16

4.2.3.33 get_value 16

4.2.3.34 get_value_without_check 16

4.2.3.35 get_zero_matrix 16

4.2.3.36 insert_array 16

4.2.3.37 insert_column_vector 16

4.2.3.38 insert_row_vector 16

4.2.3.39 insert_sub_matrix 17

4.2.3.40 insert_value 17

4.2.3.41 insert_value_without_check 17

4.2.3.42 is_matrix 17

4.2.3.43 is_non_negative_diagonal_matrix 17

4.2.3.44 is_non_negative_matrix 17

4.2.3.45 is_zero_matrix 17

4.2.3.46 largest_element_in_column_index 17

4.2.3.47 least_square 17

4.2.3.48 matlib_fabs 17

4.2.3.49 matrix_contains 17

4.2.3.50 matrix_copy 17

4.2.3.51 matrix_copy_data 17

4.2.3.52 min 18

4.2.3.53 multiply_column_with_scalar 18

4.2.3.54 multiply_matrices 18

4.2.3.55 multiply_matrices_naive 18

4.2.3.56 multiply_matrices_optimized 18

4.2.3.57 multiply_matrices_with_return 18

4.2.3.58 multiply_matrix_with_scalar 18

4.2.3.59 multiply_row_with_scalar 18

4.2.3.60 print_matrix 18

4.2.3.61 product_of_column 18

4.2.3.62 product_of_row 18

4.2.3.63 smallest_element_in_column_index 18

4.2.3.64 solve_linear 18

4.2.3.65 solve_linear_with_return 19

4.2.3.66 strassen_matrices 19

4.2.3.67 strassen_matrices_with_return 19

4.2.3.68 subtract_matrices 19

4.2.3.69 subtract_matrices_with_return 19

4.2.3.70 sum_of_column 19

4.2.3.71 sum_of_row 19

4.2.3.72 switch_rows 19
4.2.3.73 transform_to_reduced_row_echelon_form 19
4.2.3.74 transpose_matrix 19
4.2.3.75 transpose_matrix_with_return 19
4.3 TDDD77/matrixlibrary/include/sparse_lib.h File Reference 19
4.3.1 Typedef Documentation 20
4.3.1.1 sparse_matrix 20
4.3.2 Function Documentation 20
4.3.2.1 conjugate_gradient 20
4.3.2.2 copy_sparse_matrix 20
4.3.2.3 create_empty_sparse_matrix 20
4.3.2.4 create_sparse_matrix 20
4.3.2.5 free_sparse_matrix 20
4.3.2.6 matrix_sparsity 20
4.3.2.7 multiply_sparse_matrix_matrix 20
4.3.2.8 multiply_sparse_matrix_vector 21
4.3.2.9 print_sparse_matrix 21
4.3.2.10 sparse_to_normal 21
4.3.2.11 transpose_sparse_matrix 21
4.3.2.12 transpose_sparse_matrix_with_return 21
4.4 TDDD77/matrixlibrary/src/matLib.c File Reference 21
4.4.1 Function Documentation 23
4.4.1.1 add_matrices 23
4.4.1.2 add_matrices_with_return 23
4.4.1.3 add_rows 23
4.4.1.4 check_boundaries 23
4.4.1.5 compare_elements 23
4.4.1.6 compare_matrices 23
4.4.1.7 create_identity_matrix 23
4.4.1.8 create_matrix 23
4.4.1.9 create_zero_matrix 23
4.4.1.10 crout 23
4.4.1.11 derivate_matrix_with_return 23
4.4.1.12 divide_column_with_scalar 23
4.4.1.13 divide_matrix_with_scalar 24
4.4.1.14 divide_row_with_scalar 24
4.4.1.15 dot_product 24
4.4.1.16 first_nonzero_in_column_index 24
4.4.1.17 first_nonzero_in_row_index 24
4.4.1.18 forward_backward 24

4.4.1.19 free_matrix 24
4.4.1.20 gauss_jordan 24
4.4.1.21 gauss_jordan_solver 24
4.4.1.22 get_column_vector 24
4.4.1.23 get_column_vector_with_return 24
4.4.1.24 get_determinant 24
4.4.1.25 get_diagonal 25
4.4.1.26 get_inverse 25
4.4.1.27 get_inverse_of_2x2 25
4.4.1.28 get_inverse_of_2x2_with_return 25
4.4.1.29 get_matrix_with_only_pivots 25
4.4.1.30 get_row_vector 25
4.4.1.31 get_row_vector_with_return 25
4.4.1.32 get_sub_matrix 25
4.4.1.33 get_value 25
4.4.1.34 get_value_without_check 25
4.4.1.35 get_zero_matrix 25
4.4.1.36 insert_array 25
4.4.1.37 insert_column_vector 25
4.4.1.38 insert_row_vector 26
4.4.1.39 insert_sub_matrix 26
4.4.1.40 insert_value 26
4.4.1.41 insert_value_without_check 26
4.4.1.42 is_matrix 26
4.4.1.43 is_non_negative_diagonal_matrix 26
4.4.1.44 is_non_negative_matrix 26
4.4.1.45 is_zero_matrix 26
4.4.1.46 largest_element_in_column_index 26
4.4.1.47 least_square 26
4.4.1.48 matlib_fabs 26
4.4.1.49 matrix_contains 26
4.4.1.50 matrix_copy 26
4.4.1.51 matrix_copy_data 27
4.4.1.52 min 27
4.4.1.53 multiply_column_with_scalar 27
4.4.1.54 multiply_matrices 27
4.4.1.55 multiply_matrices_naive 27
4.4.1.56 multiply_matrices_optimized 27
4.4.1.57 multiply_matrices_with_return 27
4.4.1.58 multiply_matrix_with_scalar 27

4.4.1.59 multiply_row_with_scalar 27
4.4.1.60 print_matrix 27
4.4.1.61 product_of_column 27
4.4.1.62 product_of_row 27
4.4.1.63 smallest_element_in_column_index 27
4.4.1.64 solve_linear 28
4.4.1.65 solve_linear_with_return 28
4.4.1.66 strassen_matrices 28
4.4.1.67 strassen_matrices_with_return 28
4.4.1.68 subtract_matrices 28
4.4.1.69 subtract_matrices_with_return 28
4.4.1.70 sum_of_column 28
4.4.1.71 sum_of_row 28
4.4.1.72 switch_rows 28
4.4.1.73 transform_to_reduced_row_echelon_form 28
4.4.1.74 transpose_matrix 28
4.4.1.75 transpose_matrix_with_return 28
4.5 TDDD77/matrixlibrary/src/sparse_lib.c File Reference 28
4.5.1 Function Documentation 29
4.5.1.1 conjugate_gradient 29
4.5.1.2 copy_sparse_matrix 29
4.5.1.3 create_empty_sparse_matrix 29
4.5.1.4 create_sparse_matrix 29
4.5.1.5 free_sparse_matrix 29
4.5.1.6 matrix_sparsity 29
4.5.1.7 multiply_sparse_matrix_matrix 29
4.5.1.8 multiply_sparse_matrix_vector 29
4.5.1.9 print_sparse_matrix 30
4.5.1.10 sparse_to_normal 30
4.5.1.11 transpose_sparse_matrix 30
4.5.1.12 transpose_sparse_matrix_with_return 30
4.6 TDDD77/quadopt/include/problem.h File Reference 30
4.6.1 Typedef Documentation 31
4.6.1.1 problem 31
4.6.2 Function Documentation 31
4.6.2.1 create_problem 31
4.6.2.2 free_problem 31
4.6.2.3 get_active_conditions 31
4.6.2.4 get_active_conditions_rhs 31
4.6.2.5 get_solution_value 31

4.6.2.6 get_sparse_active_conditions 31
4.6.2.7 is_feasible_point 31
4.6.2.8 print_problem 31
4.6.2.9 print_solution 31
4.6.2.10 time_to_exit 31
4.7 TDDD77/quadopt/include/simplex.h File Reference 31
4.7.1 Function Documentation 32
4.7.1.1 simplex_phase_1 32
4.8 TDDD77/quadopt/include/solver.h File Reference 32
4.8.1 Function Documentation 32
4.8.1.1 quadopt_solver 32
4.8.1.2 remove_constraint 32
4.9 TDDD77/quadopt/include/subproblem.h File Reference 32
4.9.1 Function Documentation 32
4.9.1.1 solve_subproblem 32
4.10 TDDD77/quadopt/include/trans_con.h File Reference 32
4.10.1 Function Documentation 33
4.10.1.1 create_objective 33
4.10.1.2 trans_dyn_cons 33
4.10.1.3 trans_ineq_cons 33
4.11 TDDD77/quadopt/include/work_set.h File Reference 33
4.11.1 Typedef Documentation 33
4.11.1.1 work_set 33
4.11.2 Function Documentation 33
4.11.2.1 work_set_append 33
4.11.2.2 work_set_clear 34
4.11.2.3 work_set_contains 34
4.11.2.4 work_set_create 34
4.11.2.5 work_set_free 34
4.11.2.6 work_set_print 34
4.11.2.7 work_set_remove 34
4.12 TDDD77/quadopt/src/problem.c File Reference 34
4.12.1 Function Documentation 34
4.12.1.1 create_problem 34
4.12.1.2 fill_constraint_matrices 35
4.12.1.3 free_problem 35
4.12.1.4 get_active_conditions 35
4.12.1.5 get_active_conditions_rhs 35
4.12.1.6 get_solution_value 35
4.12.1.7 get_sparse_active_conditions 35

4.12.1.8 is_feasible_point 35
4.12.1.9 print_problem 35
4.12.1.10 print_solution 35
4.12.1.11 time_to_exit 35
4.13 TDDD77/quadopt/src/simplex.c File Reference 35
4.13.1 Function Documentation 36
4.13.1.1 convert_geq_to_leq 36
4.13.1.2 create_basis 36
4.13.1.3 insert_constraints 36
4.13.1.4 insert_objective_function 36
4.13.1.5 insert_simplex_variables 36
4.13.1.6 is_neg_tableau_row 36
4.13.1.7 min_test 36
4.13.1.8 neg_equality 36
4.13.1.9 remove_variables 36
4.13.1.10 set_variables 36
4.13.1.11 simplex_min 36
4.13.1.12 simplex_phase_1 36
4.13.1.13 split_eq_variables 36
4.13.1.14 split_ineq_variables 36
4.14 TDDD77/quadopt/src/solver.c File Reference 36
4.14.1 Function Documentation 37
4.14.1.1 copy_solution 37
4.14.1.2 fill_active_set 37
4.14.1.3 prefill_set 37
4.14.1.4 quadopt_solver 37
4.14.1.5 remove_constraint 37
4.14.1.6 take_step 37
4.15 TDDD77/quadopt/src/subproblem.c File Reference 37
4.15.1 Function Documentation 37
4.15.1.1 KKT_sub 37
4.15.1.2 KKT_sub_sparse 37
4.15.1.3 range_space 37
4.15.1.4 range_space_sparse 37
4.15.1.5 solve_subproblem 37
4.16 TDDD77/quadopt/src/trans_con.c File Reference 37
4.16.1 Function Documentation 38
4.16.1.1 create_objective 38
4.16.1.2 fix_g 38
4.16.1.3 insert_A_matrices 38

4.16.1.4 insert_B_matrices 38
4.16.1.5 insert_fx 38
4.16.1.6 insert_identity_matrices 38
4.16.1.7 insert_u_identity_matrices 38
4.16.1.8 insert_x_identity_matrices 38
4.16.1.9 trans_dyn_cons 38
4.16.1.10 trans_ineq_cons 38
4.17 TDDD77/quadopt/src/work_set.c File Reference 38
4.17.1 Function Documentation 39
4.17.1.1 work_set_append 39
4.17.1.2 work_set_clear 39
4.17.1.3 work_set_contains 39
4.17.1.4 work_set_create 39
4.17.1.5 work_set_free 39
4.17.1.6 work_set_print 39
4.17.1.7 work_set_remove 39

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

matrix 5
problem 5
sparse_matrix 9
work_set 9

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

TDDD77/matlab/quadopt.c 11
TDDD77/matrixlibrary/include/matLib.h 11
TDDD77/matrixlibrary/include/sparse_lib.h 19
TDDD77/matrixlibrary/src/matLib.c 21
TDDD77/matrixlibrary/src/sparse_lib.c 28
TDDD77/quadopt/include/problem.h 30
TDDD77/quadopt/include/simplex.h 31
TDDD77/quadopt/include/solver.h 32
TDDD77/quadopt/include/subproblem.h 32
TDDD77/quadopt/include/trans_con.h 32
TDDD77/quadopt/include/work_set.h 33
TDDD77/quadopt/src/problem.c 34
TDDD77/quadopt/src/simplex.c 35
TDDD77/quadopt/src/solver.c 36
TDDD77/quadopt/src/subproblem.c 37
TDDD77/quadopt/src/trans_con.c 37
TDDD77/quadopt/src/work_set.c 38

Chapter 3

Data Structure Documentation

3.1 matrix Struct Reference

```
#include <matLib.h>
```

Data Fields

- size_t columns
- size_t rows
- size_t size
- value ∗ start
- bool diagonals

3.1.1 Detailed Description

Uncomment to allow parallel operations This is the core-struct in this library. All matrix-operations are based on this Struct.

3.1.2 Field Documentation

3.1.2.1 size_t columns

3.1.2.2 bool diagonals

3.1.2.3 size_t rows

3.1.2.4 size_t size

3.1.2.5 value∗ start

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

- TDDD77/matrixlibrary/include/matLib.h

3.2 problem Struct Reference

```
#include <problem.h>
```

Data Fields

- matrix ∗ Q
- matrix ∗ Q_inv
- sparse_matrix ∗ sparse_Q
- sparse_matrix ∗ sparse_Q_inv
- bool is_sparse
- matrix ∗ q
- size_t variable_count
- size_t equality_count
- matrix ∗ E
- sparse_matrix ∗ sparse_E
- matrix ∗ h
- size_t inequality_count
- matrix ∗ F
- sparse_matrix ∗ sparse_F
- matrix ∗ g
- matrix ∗ A
- sparse_matrix ∗∗ sparse_A
- matrix ∗ b
- size_t constraints_count
- bool has_start_point
- matrix ∗ z0
- matrix ∗ z
- matrix ∗ solution
- value solution_value
- bool has_solution
- matrix ∗ p
- matrix ∗ gk
- value step
- matrix ∗ lagrange
- work_set ∗ active_set
- value accuracy
- int max_iter
- int max_micro_sec
- bool check_time

3.2.1 Detailed Description

Allocates the problem and sets all necessary variables

3.2.2 Field Documentation

3.2.2.1 matrix∗ A

All constraints left-hand side coefficients.

3.2.2.2 value accuracy

3.2.2.3 work_set∗ active_set

The active constraints.

3.2.2.4 matrix∗ b

All constraints right-hand side constraints.

3.2.2.5 bool check_time

3.2.2.6 size_t constraints_count

Total number of constraints.

3.2.2.7 matrix∗ E

Equality constraints left-hand side coefficient.

3.2.2.8 size_t equality_count

Number of equality constraints (Rows in the equality constraints matrices).

3.2.2.9 matrix∗ F

Larger-than constraints left-hand side coefficient.

3.2.2.10 matrix∗ g

Larger-than constraints right-hand side constraint.

3.2.2.11 matrix∗ gk

gk = Qz + q, help matrix for the subproblem.

See also

Q
z
q

3.2.2.12 matrix∗ h

Equality constraints right-hand side constraint.

3.2.2.13 bool has_solution

3.2.2.14 bool has_start_point

3.2.2.15 size_t inequality_count

Number of larger-than constraints (Rows in the larger-than constraints matrices).

3.2.2.16 bool is_sparse

3.2.2.17 matrix∗ lagrange

The lagrange multipliers.

3.2.2.18 int max_iter

3.2.2.19 int max_micro_sec

3.2.2.20 matrix∗ p

Current step direction towards the solution.

3.2.2.21 matrix∗ Q

The matrix containing the quadratic optimization problem.

3.2.2.22 matrix∗ q

The matrix containing the linear optimization problem.

3.2.2.23 matrix∗ Q_inv

Q inverse.

3.2.2.24 matrix∗ solution

The final point in the solution.

3.2.2.25 value solution_value

The value of the solution point.

3.2.2.26 sparse_matrix∗∗ sparse_A

3.2.2.27 sparse_matrix∗ sparse_E

3.2.2.28 sparse_matrix∗ sparse_F

3.2.2.29 sparse_matrix∗ sparse_Q

3.2.2.30 sparse_matrix∗ sparse_Q_inv

3.2.2.31 value step

How far we will step towards the solution.

3.2.2.32 size_t variable_count

The number of variables in the problem.

3.2.2.33 matrix∗ z

The current point in the solution.

3.2.2.34 matrix∗ z0

The starting point for the solution.

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

- TDDD77/quadopt/include/problem.h

3.3 sparse_matrix Struct Reference

```
#include <sparse_lib.h>
```

Data Fields

- size_t size
- size_t rows
- size_t columns
- value ∗ A
- size_t ∗ rA
- size_t ∗ cA

3.3.1 Detailed Description

Store sparse matrix using COO (coordinate list)

3.3.2 Field Documentation

3.3.2.1 value∗ A

3.3.2.2 size_t∗ cA

3.3.2.3 size_t columns

3.3.2.4 size_t∗ rA

3.3.2.5 size_t rows

3.3.2.6 size_t size

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

- TDDD77/matrixlibrary/include/sparse_lib.h

3.4 work_set Struct Reference

```
#include <work_set.h>
```

Data Fields

- size_t max_count
- size_t count
- size_t ∗ data

3.4.1 Detailed Description

Structure for storing different sets

3.4.2 Field Documentation

3.4.2.1 size_t count

Number of elements in the work set.

3.4.2.2 size_t∗ data

Array of elements in the work set.

3.4.2.3 size_t max_count

Maximum number of elements in the work set

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

- TDDD77/quadopt/include/work_set.h

Chapter 4

File Documentation

4.1 TDDD77/matlab/quadopt.c File Reference

```
#include "mex.h"
#include "../quadopt/include/solver.h"
#include "../quadopt/include/problem.h"
```

Functions

- void mexFunction (int nlhs, mxArray ∗plhs[ ], int nrhs, const mxArray ∗prhs[ ])

4.1.1 Function Documentation

4.1.1.1 void mexFunction (int *nlhs,* mxArray ∗ *plhs[ ],* int *nrhs,* const mxArray ∗ *prhs[ ]*)

This functions creates an interface between MATLAB and the solver together with the matrixlibrary. It also converts MATLAB structured matrices into the matrixlibrary structure.

4.2 TDDD77/matrixlibrary/include/matLib.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
```

Data Structures

- struct matrix

Macros

- #define DOUBLE
- #define FORMAT_STRING "%f "
- #define PRECISION 0.0001

Typedefs

- typedef double value
- typedef struct matrix matrix

Functions

- matrix ∗ create_matrix (size_t row, size_t col)
- matrix ∗ create_zero_matrix (size_t row, size_t col)
- matrix ∗ create_identity_matrix (size_t row, size_t col)
- value dot_product (matrix ∗r, matrix ∗v)
- void free_matrix (matrix ∗mat)
- void print_matrix (matrix ∗mat)
- bool check_boundaries (size_t row, size_t col, matrix ∗mat)
- bool insert_array (value arr[ ], matrix ∗mat)
- bool compare_matrices (matrix ∗a, matrix ∗b)
- bool is_matrix (matrix ∗a, matrix ∗b)
- bool insert_value (value insert, size_t row, size_t col, matrix ∗mat)
- void insert_value_without_check (value insert, size_t row, size_t col, matrix ∗mat)
- value get_value (size_t row, size_t col, matrix ∗mat)
- value get_value_without_check (size_t row, size_t col, matrix ∗mat)
- bool add_matrices (matrix ∗a, matrix ∗b, matrix ∗c)
- matrix ∗ add_matrices_with_return (matrix ∗a, matrix ∗b)
- bool subtract_matrices (matrix ∗a, matrix ∗b, matrix ∗c)
- matrix ∗ subtract_matrices_with_return (matrix ∗a, matrix ∗b)
- bool multiply_matrices (matrix ∗a, matrix ∗b, matrix ∗c)
- bool multiply_matrices_naive (matrix ∗a, matrix ∗b, matrix ∗c)
- bool multiply_matrices_optimized (matrix ∗a, matrix ∗b, matrix ∗c)
- matrix ∗ strassen_matrices_with_return (matrix ∗a, matrix ∗b)
- bool strassen_matrices (matrix ∗a, matrix ∗b, matrix ∗c)
- matrix ∗ multiply_matrices_with_return (matrix ∗a, matrix ∗b)
- value get_determinant (matrix ∗a)
- bool get_inverse (matrix ∗a, matrix ∗c)
- matrix ∗ get_inverse_of_2x2_with_return (matrix ∗a)
- bool get_inverse_of_2x2 (matrix ∗a, matrix ∗b)
- bool solve_linear (matrix ∗a, matrix ∗x, matrix ∗b)
- matrix ∗ solve_linear_with_return (matrix ∗a, matrix ∗b)
- bool crout (matrix ∗a, matrix ∗l, matrix ∗u)
- void forward_backward (matrix ∗l, matrix ∗u, matrix ∗x, matrix ∗b)
- void least_square (matrix ∗a, matrix ∗x, matrix ∗b)
- bool gauss_jordan (matrix ∗a)
- bool gauss_jordan_solver (matrix ∗a, matrix ∗x, matrix ∗b)
- matrix ∗ get_matrix_with_only_pivots (matrix ∗a)
- value min (value a, value b)
- size_t largest_element_in_column_index (size_t column, size_t start, matrix ∗a)
- size_t smallest_element_in_column_index (size_t column, size_t start, matrix ∗a)
- size_t first_nonezero_in_column_index (size_t column, size_t start, matrix ∗a)
- size_t first_nonezero_in_row_index (size_t row, size_t start, matrix ∗a)
- void add_rows (size_t row1, size_t row2, matrix ∗a)
- bool transpose_matrix (matrix ∗a, matrix ∗b)
- matrix ∗ transpose_matrix_with_return (matrix ∗a)
- value sum_of_row (size_t row, matrix ∗mat)
- value sum_of_column (size_t column, matrix ∗mat)
- value product_of_row (size_t row, matrix ∗mat)

- value product_of_column (size_t column, matrix ∗mat)
- void multiply_matrix_with_scalar (value scal, matrix ∗mat)
- void divide_matrix_with_scalar (value scal, matrix ∗mat)
- void multiply_row_with_scalar (value scal, size_t row, matrix ∗mat)
- void divide_row_with_scalar (value scal, size_t row, matrix ∗mat)
- void multiply_column_with_scalar (value scal, size_t col, matrix ∗mat)
- void divide_column_with_scalar (value scal, size_t col, matrix ∗mat)
- bool get_row_vector (size_t row, matrix ∗a, matrix ∗b)
- matrix ∗ get_row_vector_with_return (size_t row, matrix ∗a)
- bool insert_row_vector (size_t row, matrix ∗a, matrix ∗b)
- bool switch_rows (size_t row1, size_t row2, matrix ∗a)
- bool get_column_vector (size_t column, matrix ∗a, matrix ∗b)
- matrix ∗ get_column_vector_with_return (size_t column, matrix ∗a)
- bool insert_column_vector (size_t column, matrix ∗a, matrix ∗b)
- bool get_sub_matrix (size_t start_row, size_t end_row, size_t start_col, size_t end_col, matrix ∗a, matrix ∗b)
- bool insert_sub_matrix (size_t start_row, size_t end_row, size_t start_col, size_t end_col, matrix ∗b, matrix ∗a)
- matrix ∗ matrix_copy (matrix ∗source)
- void matrix_copy_data (matrix ∗A, matrix ∗B)
- bool is_zero_matrix (matrix ∗v)
- bool is_non_negative_matrix (matrix ∗v)
- bool is_non_negative_diagonal_matrix (matrix ∗A)
- bool get_diagonal (matrix ∗a, matrix ∗b)
- matrix ∗ derivate_matrix_with_return (size_t var, matrix ∗a)
- void transform_to_reduced_row_echelon_form (matrix ∗M)
- bool matrix_contains (value a, matrix ∗b)
- int compare_elements (value a, value b)
- matrix ∗ get_zero_matrix (size_t rows, size_t columns)
- value matlib_fabs (value a)

4.2.1 Macro Definition Documentation

4.2.1.1 #define DOUBLE

Only standardlibraries Uncomment which mode you want the library to run in

4.2.1.2 #define FORMAT_STRING "%f "

4.2.1.3 #define PRECISION 0.0001

4.2.2 Typedef Documentation

4.2.2.1 typedef struct matrix matrix

4.2.2.2 typedef double value

Setup for the preprocessor depending on mode

4.2.3 Function Documentation

4.2.3.1 bool add_matrices (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

Adds a and b into c

4.2.3.2 matrix∗ add_matrices_with_return (matrix ∗ *a*, matrix ∗ *b*)

Adds a and b by returning a pointer a matrix with a+b

4.2.3.3 void add_rows (size_t *row1*, size_t *row2*, matrix ∗ *a*)

Adds each element in row1 and row 2 and puts the result on row2

4.2.3.4 bool check_boundaries (size_t *row*, size_t *col*, matrix ∗ *mat*)

Checks if the position exists in the matrix

4.2.3.5 int compare_elements (value *a*, value *b*)

Compare two element values

4.2.3.6 bool compare_matrices (matrix ∗ *a*, matrix ∗ *b*)

Returns true if matrices a and b look the same

4.2.3.7 matrix∗ create_identity_matrix (size_t *row*, size_t *col*)

Creates a identity matrix

4.2.3.8 matrix∗ create_matrix (size_t *row*, size_t *col*)

Create a matrix

4.2.3.9 matrix∗ create_zero_matrix (size_t *row*, size_t *col*)

Is normally not needed for this implementation but might be needed on others

4.2.3.10 bool crout (matrix ∗ *a*, matrix ∗ *l*, matrix ∗ *u*)

Crout algorithm to divide matrix a into l and u that holds a=lu

4.2.3.11 matrix∗ derivate_matrix_with_return (size_t *var*, matrix ∗ *a*)

Returns a pointer to a matrix with the derivative of var if the a matrix second order coefficiants

4.2.3.12 void divide_column_with_scalar (value *scal*, size_t *col*, matrix ∗ *mat*)

Divides a column with a scalar

4.2.3.13 void divide_matrix_with_scalar (value *scal*, matrix ∗ *mat*)

Divides matrix mat with scalar

4.2.3.14 void divide_row_with_scalar (value *scal,* size_t *row,* matrix ∗ *mat*)

Divides a row with a scalar

4.2.3.15 value dot_product (matrix ∗ *r,* matrix ∗ *v*)

Calculate the dot product

4.2.3.16 size_t first_nonezero_in_column_index (size_t *column,* size_t *start,* matrix ∗ *a*)

Returns on which row the first nonezero element is in the column is after start returns -1 if no nonezero element is found

4.2.3.17 size_t first_nonezero_in_row_index (size_t *row,* size_t *start,* matrix ∗ *a*)

Returns on which column the first nonezero element is in the column is after start returns -1 if no nonezero element is found

4.2.3.18 void forward_backward (matrix ∗ *l,* matrix ∗ *u,* matrix ∗ *x,* matrix ∗ *b*)

Solves lux=b using backward and forward substitution

4.2.3.19 void free_matrix (matrix ∗ *mat*)

Destroy a matrix

4.2.3.20 bool gauss_jordan (matrix ∗ *a*)

Gauss eliminates the matrix a

4.2.3.21 bool gauss_jordan_solver (matrix ∗ *a,* matrix ∗ *x,* matrix ∗ *b*)

Solves the system of linear equations using gauss jordan

4.2.3.22 bool get_column_vector (size_t *column,* matrix ∗ *a,* matrix ∗ *b*)

Takes column vector from matrix a and puts it into b

4.2.3.23 matrix∗ get_column_vector_with_return (size_t *column,* matrix ∗ *a*)

Takes column vector from matrix a and return a pointer to the row vector

4.2.3.24 value get_determinant (matrix ∗ *a*)

Returns the determinant of matrix a

4.2.3.25 bool get_diagonal (matrix ∗ *a,* matrix ∗ *b*)

Takes the diagonal in a and puts it into b

4.2.3.26 bool get_inverse (**matrix** ∗ *a,* **matrix** ∗ *c*)

Calculates the inverse of a and puts it into c

4.2.3.27 bool get_inverse_of_2x2 (**matrix** ∗ *a,* **matrix** ∗ *b*)

4.2.3.28 **matrix**∗ get_inverse_of_2x2_with_return (**matrix** ∗ *a*)

4.2.3.29 **matrix**∗ get_matrix_with_only_pivots (**matrix** ∗ *a*)

Returns a matrix with only pivots elements from a

4.2.3.30 bool get_row_vector (size_t *row,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Takes row vector from matrix a and puts it into b

4.2.3.31 **matrix**∗ get_row_vector_with_return (size_t *row,* **matrix** ∗ *a*)

Returns row vector row from matrix a with a pointer to a matrix

4.2.3.32 bool get_sub_matrix (size_t *start_row,* size_t *end_row,* size_t *start_col,* size_t *end_col,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Get a sub matrix from a

4.2.3.33 **value** get_value (size_t *row,* size_t *col,* **matrix** ∗ *mat*)

Get a value from matrix

4.2.3.34 **value** get_value_without_check (size_t *row,* size_t *col,* **matrix** ∗ *mat*)

As get_value without check

4.2.3.35 **matrix**∗ get_zero_matrix (size_t *rows,* size_t *columns*)

Creates new matrix with zero values

4.2.3.36 bool insert_array (**value** *arr[ ],* **matrix** ∗ *mat*)

Insert a array into the matrix

4.2.3.37 bool insert_column_vector (size_t *column,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Inserts column vector a into matrix b at position column

4.2.3.38 bool insert_row_vector (size_t *row,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Inserts row vector a into b:s row

4.2.3.39 bool insert_sub_matrix (size_t *start_row*, size_t *end_row*, size_t *start_col*, size_t *end_col*, matrix ∗ *b*, matrix ∗ *a*)

4.2.3.40 bool insert_value (value *insert*, size_t *row*, size_t *col*, matrix ∗ *mat*)

Insert a value into matrix

4.2.3.41 void insert_value_without_check (value *insert*, size_t *row*, size_t *col*, matrix ∗ *mat*)

As insert_value without check

4.2.3.42 bool is_matrix (matrix ∗ *a*, matrix ∗ *b*)

Return true if the matrix are the same

4.2.3.43 bool is_non_negative_diagonal_matrix (matrix ∗ *A*)

Checks if all elements along the diagonal in a symmetric matrix is positive

4.2.3.44 bool is_non_negative_matrix (matrix ∗ *v*)

Checks if all elements in a matrix is positive

4.2.3.45 bool is_zero_matrix (matrix ∗ *v*)

Checks if all elements in a matrix is equal to zero

4.2.3.46 size_t largest_element_in_column_index (size_t *column*, size_t *start*, matrix ∗ *a*)

Returns on which row the largest element in the column is after start

4.2.3.47 void least_square (matrix ∗ *a*, matrix ∗ *x*, matrix ∗ *b*)

If no solution can be found with solve_linear, this function finds the closest one

4.2.3.48 value matlib_fabs (value *a*)

Returns the absolute value of a

4.2.3.49 bool matrix_contains (value *a*, matrix ∗ *b*)

Return true if b contains value a

4.2.3.50 matrix∗ matrix_copy (matrix ∗ *source*)

Copy and return new matrix.

4.2.3.51 void matrix_copy_data (matrix ∗ *A*, matrix ∗ *B*)

Copies all the data from matrix A into matrix B

4.2.3.52 value min (value *a*, value *b*)

Returns the lowest of the two values

4.2.3.53 void multiply_column_with_scalar (value *scal*, size_t *col*, matrix ∗ *mat*)

Multiplies a column with a scalar

4.2.3.54 bool multiply_matrices (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

Multiply a and b into c. c=a∗b

4.2.3.55 bool multiply_matrices_naive (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

4.2.3.56 bool multiply_matrices_optimized (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

4.2.3.57 matrix∗ multiply_matrices_with_return (matrix ∗ *a*, matrix ∗ *b*)

Multiply a and b by returning a pointer to a new matrix with a∗b

4.2.3.58 void multiply_matrix_with_scalar (value *scal*, matrix ∗ *mat*)

Multiplies matrix mat with scalar

4.2.3.59 void multiply_row_with_scalar (value *scal*, size_t *row*, matrix ∗ *mat*)

Multiplies a row with a scalar

4.2.3.60 void print_matrix (matrix ∗ *mat*)

Prints the matrix

4.2.3.61 value product_of_column (size_t *column*, matrix ∗ *mat*)

Return the product of a column in matrix mat

4.2.3.62 value product_of_row (size_t *row*, matrix ∗ *mat*)

Return the product of a row in matrix mat

4.2.3.63 size_t smallest_element_in_column_index (size_t *column*, size_t *start*, matrix ∗ *a*)

Returns on which row the smallest element in the column is after start

4.2.3.64 bool solve_linear (matrix ∗ *a*, matrix ∗ *x*, matrix ∗ *b*)

Solves Ax=B

4.2.3.65 matrix∗ solve_linear_with_return (matrix ∗ *a*, matrix ∗ *b*)

Solves ax=b by returning a pointer to x

4.2.3.66 bool strassen_matrices (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

4.2.3.67 matrix∗ strassen_matrices_with_return (matrix ∗ *a*, matrix ∗ *b*)

4.2.3.68 bool subtract_matrices (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

Subtract a and b into c. c=a-b

4.2.3.69 matrix∗ subtract_matrices_with_return (matrix ∗ *a*, matrix ∗ *b*)

Subtracts a and b by returning a pointer a matrix with a-b

4.2.3.70 value sum_of_column (size_t *column*, matrix ∗ *mat*)

Return the sum of a column in matrix mat

4.2.3.71 value sum_of_row (size_t *row*, matrix ∗ *mat*)

Return the sum of a row in matrix mat

4.2.3.72 bool switch_rows (size_t *row1*, size_t *row2*, matrix ∗ *a*)

Switches rows in a

4.2.3.73 void transform_to_reduced_row_echelon_form (matrix ∗ *M*)

4.2.3.74 bool transpose_matrix (matrix ∗ *a*, matrix ∗ *b*)

Transposes matrix a into b

4.2.3.75 matrix∗ transpose_matrix_with_return (matrix ∗ *a*)

Transposes matrix a by returning a pointer to a:s transpose

4.3 TDDD77/matrixlibrary/include/sparse_lib.h File Reference

```
#include <matLib.h>
```

Data Structures

- struct sparse_matrix

Typedefs

- typedef struct sparse_matrix sparse_matrix

Functions

- sparse_matrix ∗ create_sparse_matrix (matrix ∗Ain, int size)
- sparse_matrix ∗ create_empty_sparse_matrix (size_t size)
- matrix ∗ sparse_to_normal (sparse_matrix ∗S)
- size_t matrix_sparsity (matrix ∗A)
- bool multiply_sparse_matrix_vector (sparse_matrix ∗A, matrix ∗x, matrix ∗Ax)
- matrix ∗ multiply_sparse_matrix_matrix (sparse_matrix ∗A, matrix ∗B)
- sparse_matrix ∗ copy_sparse_matrix (sparse_matrix ∗Ain)
- void transpose_sparse_matrix (sparse_matrix ∗Ain)
- sparse_matrix ∗ transpose_sparse_matrix_with_return (sparse_matrix ∗Ain)
- void print_sparse_matrix (sparse_matrix ∗S)
- void free_sparse_matrix (sparse_matrix ∗S)
- bool conjugate_gradient (sparse_matrix ∗A, matrix ∗x, matrix ∗b)

4.3.1 Typedef Documentation

4.3.1.1 typedef struct **sparse_matrix sparse_matrix**

4.3.2 Function Documentation

4.3.2.1 bool conjugate_gradient (**sparse_matrix** ∗ *A*, **matrix** ∗ *x*, **matrix** ∗ *b*)

Solves Ax = b, x should be set to 0 Is not used, due to not working with MATLAB gate.

4.3.2.2 **sparse_matrix**∗ copy_sparse_matrix (**sparse_matrix** ∗ *Ain*)

Copies a sparse matrix and returns it

4.3.2.3 **sparse_matrix**∗ create_empty_sparse_matrix (size_t *size*)

Creates an empty sparse matrix

4.3.2.4 **sparse_matrix**∗ create_sparse_matrix (**matrix** ∗ *Ain*, int *size*)

Creates a sparse matrix out of a normal matrix

4.3.2.5 void free_sparse_matrix (**sparse_matrix** ∗ *S*)

Frees allocated memory of the sparse matrix

4.3.2.6 size_t matrix_sparsity (**matrix** ∗ *A*)

Returns number of elements != 0

4.3.2.7 **matrix**∗ multiply_sparse_matrix_matrix (**sparse_matrix** ∗ *A*, **matrix** ∗ *B*)

Multiplies sparse matrix with a normal matrix. Returns a normal matrix.

4.3.2.8 bool multiply_sparse_matrix_vector (**sparse_matrix** ∗ *A,* **matrix** ∗ *x,* **matrix** ∗ *Ax*)

Multiplies sparse matrix with normal vector, stores result in normal matrix Ax

4.3.2.9 void print_sparse_matrix (**sparse_matrix** ∗ *S*)

Prints sparse matrix

4.3.2.10 **matrix**∗ sparse_to_normal (**sparse_matrix** ∗ *S*)

Converts sparse matrix to normal matrix

4.3.2.11 void transpose_sparse_matrix (**sparse_matrix** ∗ *Ain*)

Transposes input sparse matrix

4.3.2.12 **sparse_matrix**∗ transpose_sparse_matrix_with_return (**sparse_matrix** ∗ *Ain*)

Transposes a sparse matrix and returns it in a new sparse matrix

4.4 TDDD77/matrixlibrary/src/matLib.c File Reference

```
#include <matLib.h>
```

Functions

- matrix ∗ create_matrix (size_t row, size_t col)
- matrix ∗ create_zero_matrix (size_t row, size_t col)
- matrix ∗ create_identity_matrix (size_t row, size_t col)
- void free_matrix (matrix ∗mat)
- value dot_product (matrix ∗r, matrix ∗v)
- void print_matrix (matrix ∗mat)
- bool check_boundaries (size_t row, size_t col, matrix ∗mat)
- bool insert_array (value arr[ ], matrix ∗mat)
- bool compare_matrices (matrix ∗a, matrix ∗b)
- bool is_matrix (matrix ∗a, matrix ∗b)
- bool insert_value (value insert, size_t row, size_t col, matrix ∗mat)
- void insert_value_without_check (value insert, size_t row, size_t col, matrix ∗mat)
- value get_value (size_t row, size_t col, matrix ∗mat)
- value get_value_without_check (size_t row, size_t col, matrix ∗mat)
- bool add_matrices (matrix ∗a, matrix ∗b, matrix ∗c)
- matrix ∗ add_matrices_with_return (matrix ∗a, matrix ∗b)
- bool subtract_matrices (matrix ∗a, matrix ∗b, matrix ∗c)
- matrix ∗ subtract_matrices_with_return (matrix ∗a, matrix ∗b)
- bool multiply_matrices (matrix ∗a, matrix ∗b, matrix ∗c)
- bool multiply_matrices_naive (matrix ∗a, matrix ∗b, matrix ∗c)
- bool multiply_matrices_optimized (matrix ∗a, matrix ∗b, matrix ∗c)
- matrix ∗ strassen_matrices_with_return (matrix ∗a, matrix ∗b)
- bool strassen_matrices (matrix ∗a, matrix ∗b, matrix ∗c)
- matrix ∗ multiply_matrices_with_return (matrix ∗a, matrix ∗b)

- value get_determinant (matrix ∗a)
- bool get_inverse (matrix ∗a, matrix ∗c)
- matrix ∗ get_inverse_of_2x2_with_return (matrix ∗a)
- bool get_inverse_of_2x2 (matrix ∗a, matrix ∗b)
- bool solve_linear (matrix ∗a, matrix ∗x, matrix ∗b)
- matrix ∗ solve_linear_with_return (matrix ∗a, matrix ∗b)
- bool crout (matrix ∗a, matrix ∗l, matrix ∗u)
- void forward_backward (matrix ∗l, matrix ∗u, matrix ∗x, matrix ∗b)
- void least_square (matrix ∗a, matrix ∗x, matrix ∗b)
- bool gauss_jordan (matrix ∗a)
- bool gauss_jordan_solver (matrix ∗a, matrix ∗x, matrix ∗b)
- matrix ∗ get_matrix_with_only_pivots (matrix ∗a)
- value min (value a, value b)
- size_t largest_element_in_column_index (size_t column, size_t start, matrix ∗a)
- size_t smallest_element_in_column_index (size_t column, size_t start, matrix ∗a)
- size_t first_nonezero_in_column_index (size_t column, size_t start, matrix ∗a)
- size_t first_nonezero_in_row_index (size_t row, size_t start, matrix ∗a)
- void add_rows (size_t row1, size_t row2, matrix ∗a)
- bool transpose_matrix (matrix ∗a, matrix ∗b)
- matrix ∗ transpose_matrix_with_return (matrix ∗a)
- value sum_of_row (size_t row, matrix ∗mat)
- value sum_of_column (size_t column, matrix ∗mat)
- value product_of_row (size_t row, matrix ∗mat)
- value product_of_column (size_t column, matrix ∗mat)
- void multiply_matrix_with_scalar (value scal, matrix ∗mat)
- void divide_matrix_with_scalar (value scal, matrix ∗mat)
- void multiply_row_with_scalar (value scal, size_t row, matrix ∗mat)
- void divide_row_with_scalar (value scal, size_t row, matrix ∗mat)
- void multiply_column_with_scalar (value scal, size_t col, matrix ∗mat)
- void divide_column_with_scalar (value scal, size_t col, matrix ∗mat)
- bool get_row_vector (size_t row, matrix ∗a, matrix ∗b)
- matrix ∗ get_row_vector_with_return (size_t row, matrix ∗a)
- bool insert_row_vector (size_t row, matrix ∗a, matrix ∗b)
- bool switch_rows (size_t row1, size_t row2, matrix ∗a)
- bool get_column_vector (size_t column, matrix ∗a, matrix ∗b)
- matrix ∗ get_column_vector_with_return (size_t column, matrix ∗a)
- bool insert_column_vector (size_t column, matrix ∗a, matrix ∗b)
- bool get_sub_matrix (size_t start_row, size_t end_row, size_t start_col, size_t end_col, matrix ∗a, matrix ∗b)
- bool insert_sub_matrix (size_t start_row, size_t end_row, size_t start_col, size_t end_col, matrix ∗b, matrix ∗a)
- matrix ∗ matrix_copy (matrix ∗source)
- void matrix_copy_data (matrix ∗a, matrix ∗b)
- bool is_zero_matrix (matrix ∗v)
- bool is_non_negative_matrix (matrix ∗v)
- bool is_non_negative_diagonal_matrix (matrix ∗A)
- bool get_diagonal (matrix ∗a, matrix ∗b)
- matrix ∗ derivate_matrix_with_return (size_t var, matrix ∗a)
- void transform_to_reduced_row_echelon_form (matrix ∗M)
- bool matrix_contains (value a, matrix ∗b)
- int compare_elements (value a, value b)
- matrix ∗ get_zero_matrix (size_t rows, size_t columns)
- value matlib_fabs (value a)

4.4.1 Function Documentation

4.4.1.1 bool add_matrices (matrix ∗ *a,* matrix ∗ *b,* matrix ∗ *c*)

Adds a and b into c

4.4.1.2 matrix∗ add_matrices_with_return (matrix ∗ *a,* matrix ∗ *b*)

Adds a and b by returning a pointer a matrix with a+b

4.4.1.3 void add_rows (size_t *row1,* size_t *row2,* matrix ∗ *a*)

Adds each element in row1 and row 2 and puts the result on row2

4.4.1.4 bool check_boundaries (size_t *row,* size_t *col,* matrix ∗ *mat*)

Checks if the position exists in the matrix

4.4.1.5 int compare_elements (value *a,* value *b*)

Compare two element values

4.4.1.6 bool compare_matrices (matrix ∗ *a,* matrix ∗ *b*)

Returns true if matrices a and b look the same

4.4.1.7 matrix∗ create_identity_matrix (size_t *row,* size_t *col*)

Creates a identity matrix

4.4.1.8 matrix∗ create_matrix (size_t *row,* size_t *col*)

Create a matrix

4.4.1.9 matrix∗ create_zero_matrix (size_t *row,* size_t *col*)

Is normally not needed for this implementation but might be needed on others

4.4.1.10 bool crout (matrix ∗ *a,* matrix ∗ *l,* matrix ∗ *u*)

Crout algorithm to divide matrix a into l and u that holds a=lu

4.4.1.11 matrix∗ derivate_matrix_with_return (size_t *var,* matrix ∗ *a*)

Returns a pointer to a matrix with the derivative of var if the a matrix second order coefficiants

4.4.1.12 void divide_column_with_scalar (value *scal,* size_t *col,* matrix ∗ *mat*)

Divides a column with a scalar

4.4.1.13 void divide_matrix_with_scalar (value *scal,* matrix ∗ *mat*)

Divides matrix mat with scalar

4.4.1.14 void divide_row_with_scalar (value *scal,* size_t *row,* matrix ∗ *mat*)

Divides a row with a scalar

4.4.1.15 value dot_product (matrix ∗ *r,* matrix ∗ *v*)

Calculate the dot product

4.4.1.16 size_t first_nonezero_in_column_index (size_t *column,* size_t *start,* matrix ∗ *a*)

Returns on which row the first nonezero element is in the column is after start returns -1 if no nonezero element is found

4.4.1.17 size_t first_nonezero_in_row_index (size_t *row,* size_t *start,* matrix ∗ *a*)

Returns on which column the first nonezero element is in the column is after start returns -1 if no nonezero element is found

4.4.1.18 void forward_backward (matrix ∗ *l,* matrix ∗ *u,* matrix ∗ *x,* matrix ∗ *b*)

Solves lux=b using backward and forward substitution

4.4.1.19 void free_matrix (matrix ∗ *mat*)

Destroy a matrix

4.4.1.20 bool gauss_jordan (matrix ∗ *a*)

Gauss eliminates the matrix a

4.4.1.21 bool gauss_jordan_solver (matrix ∗ *a,* matrix ∗ *x,* matrix ∗ *b*)

Solves the system of linear equations using gauss jordan

4.4.1.22 bool get_column_vector (size_t *column,* matrix ∗ *a,* matrix ∗ *b*)

Takes column vector from matrix a and puts it into b

4.4.1.23 matrix∗ get_column_vector_with_return (size_t *column,* matrix ∗ *a*)

Takes column vector from matrix a and return a pointer to the row vector

4.4.1.24 value get_determinant (matrix ∗ *a*)

Returns the determinant of matrix a

4.4.1.25 bool get_diagonal (**matrix** ∗ *a,* **matrix** ∗ *b*)

Takes the diagonal in a and puts it into b

4.4.1.26 bool get_inverse (**matrix** ∗ *a,* **matrix** ∗ *c*)

Calculates the inverse of a and puts it into c

4.4.1.27 bool get_inverse_of_2x2 (**matrix** ∗ *a,* **matrix** ∗ *b*)

4.4.1.28 **matrix**∗ get_inverse_of_2x2_with_return (**matrix** ∗ *a*)

4.4.1.29 **matrix**∗ get_matrix_with_only_pivots (**matrix** ∗ *a*)

Returns a matrix with only pivots elements from a

4.4.1.30 bool get_row_vector (size_t *row,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Takes row vector from matrix a and puts it into b

4.4.1.31 **matrix**∗ get_row_vector_with_return (size_t *row,* **matrix** ∗ *a*)

Returns row vector row from matrix a with a pointer to a matrix

4.4.1.32 bool get_sub_matrix (size_t *start_row,* size_t *end_row,* size_t *start_col,* size_t *end_col,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Get a sub matrix from a

4.4.1.33 **value** get_value (size_t *row,* size_t *col,* **matrix** ∗ *mat*)

Get a value from matrix

4.4.1.34 **value** get_value_without_check (size_t *row,* size_t *col,* **matrix** ∗ *mat*)

As get_value without check

4.4.1.35 **matrix**∗ get_zero_matrix (size_t *rows,* size_t *columns*)

Creates new matrix with zero values

4.4.1.36 bool insert_array (**value** *arr[ ],* **matrix** ∗ *mat*)

Insert a array into the matrix

4.4.1.37 bool insert_column_vector (size_t *column,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Inserts column vector a into matrix b at position column

4.4.1.38 bool insert_row_vector (size_t *row,* matrix ∗ *a,* matrix ∗ *b*)

Inserts row vector a into b:s row

4.4.1.39 bool insert_sub_matrix (size_t *start_row,* size_t *end_row,* size_t *start_col,* size_t *end_col,* matrix ∗ *b,* matrix ∗ *a*)

4.4.1.40 bool insert_value (value *insert,* size_t *row,* size_t *col,* matrix ∗ *mat*)

Insert a value into matrix

4.4.1.41 void insert_value_without_check (value *insert,* size_t *row,* size_t *col,* matrix ∗ *mat*)

As insert_value without check

4.4.1.42 bool is_matrix (matrix ∗ *a,* matrix ∗ *b*)

Return true if the matrix are the same

4.4.1.43 bool is_non_negative_diagonal_matrix (matrix ∗ *A*)

Checks if all elements along the diagonal in a symmetric matrix is positive

4.4.1.44 bool is_non_negative_matrix (matrix ∗ *v*)

Checks if all elements in a matrix is positive

4.4.1.45 bool is_zero_matrix (matrix ∗ *v*)

Checks if all elements in a matrix is equal to zero

4.4.1.46 size_t largest_element_in_column_index (size_t *column,* size_t *start,* matrix ∗ *a*)

Returns on which row the largest element in the column is after start

4.4.1.47 void least_square (matrix ∗ *a,* matrix ∗ *x,* matrix ∗ *b*)

If no solution can be found with solve_linear, this function finds the closest one

4.4.1.48 value matlib_fabs (value *a*)

Returns the absolute value of a

4.4.1.49 bool matrix_contains (value *a,* matrix ∗ *b*)

Return true if b contains value a

4.4.1.50 matrix∗ matrix_copy (matrix ∗ *source*)

Copy and return new matrix.

4.4.1.51 void matrix_copy_data (matrix ∗ *A*, matrix ∗ *B*)

Copies all the data from matrix A into matrix B

4.4.1.52 value min (value *a*, value *b*)

Returns the lowest of the two values

4.4.1.53 void multiply_column_with_scalar (value *scal*, size_t *col*, matrix ∗ *mat*)

Multiplies a column with a scalar

4.4.1.54 bool multiply_matrices (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

Multiply a and b into c. c=a∗b

4.4.1.55 bool multiply_matrices_naive (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

4.4.1.56 bool multiply_matrices_optimized (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

4.4.1.57 matrix∗ multiply_matrices_with_return (matrix ∗ *a*, matrix ∗ *b*)

Multiply a and b by returning a pointer to a new matrix with a∗b

4.4.1.58 void multiply_matrix_with_scalar (value *scal*, matrix ∗ *mat*)

Multiplies matrix mat with scalar

4.4.1.59 void multiply_row_with_scalar (value *scal*, size_t *row*, matrix ∗ *mat*)

Multiplies a row with a scalar

4.4.1.60 void print_matrix (matrix ∗ *mat*)

Prints the matrix

4.4.1.61 value product_of_column (size_t *column*, matrix ∗ *mat*)

Return the product of a column in matrix mat

4.4.1.62 value product_of_row (size_t *row*, matrix ∗ *mat*)

Return the product of a row in matrix mat

4.4.1.63 size_t smallest_element_in_column_index (size_t *column*, size_t *start*, matrix ∗ *a*)

Returns on which row the smallest element in the column is after start

4.4.1.64 bool solve_linear (matrix ∗ *a*, matrix ∗ *x*, matrix ∗ *b*)

Solves Ax=B

4.4.1.65 matrix∗ solve_linear_with_return (matrix ∗ *a*, matrix ∗ *b*)

Solves ax=b by returning a pointer to x

4.4.1.66 bool strassen_matrices (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

4.4.1.67 matrix∗ strassen_matrices_with_return (matrix ∗ *a*, matrix ∗ *b*)

4.4.1.68 bool subtract_matrices (matrix ∗ *a*, matrix ∗ *b*, matrix ∗ *c*)

Subtract a and b into c. c=a-b

4.4.1.69 matrix∗ subtract_matrices_with_return (matrix ∗ *a*, matrix ∗ *b*)

Subtracts a and b by returning a pointer a matrix with a-b

4.4.1.70 value sum_of_column (size_t *column*, matrix ∗ *mat*)

Return the sum of a column in matrix mat

4.4.1.71 value sum_of_row (size_t *row*, matrix ∗ *mat*)

Return the sum of a row in matrix mat

4.4.1.72 bool switch_rows (size_t *row1*, size_t *row2*, matrix ∗ *a*)

Switches rows in a

4.4.1.73 void transform_to_reduced_row_echelon_form (matrix ∗ *M*)

4.4.1.74 bool transpose_matrix (matrix ∗ *a*, matrix ∗ *b*)

Transposes matrix a into b

4.4.1.75 matrix∗ transpose_matrix_with_return (matrix ∗ *a*)

Transposes matrix a by returning a pointer to a:s transpose

4.5 TDDD77/matrixlibrary/src/sparse_lib.c File Reference

```
#include <sparse_lib.h>
#include <math.h>
```

Functions

- sparse_matrix ∗ create_sparse_matrix (matrix ∗Ain, int size)
- sparse_matrix ∗ create_empty_sparse_matrix (size_t size)
- matrix ∗ sparse_to_normal (sparse_matrix ∗S)
- size_t matrix_sparsity (matrix ∗A)
- bool multiply_sparse_matrix_vector (sparse_matrix ∗A, matrix ∗x, matrix ∗Ax)
- matrix ∗ multiply_sparse_matrix_matrix (sparse_matrix ∗A, matrix ∗B)
- sparse_matrix ∗ copy_sparse_matrix (sparse_matrix ∗Ain)
- void transpose_sparse_matrix (sparse_matrix ∗Ain)
- sparse_matrix ∗ transpose_sparse_matrix_with_return (sparse_matrix ∗Ain)
- void print_sparse_matrix (sparse_matrix ∗S)
- void free_sparse_matrix (sparse_matrix ∗S)
- bool conjugate_gradient (sparse_matrix ∗A, matrix ∗x, matrix ∗b)

4.5.1 Function Documentation

4.5.1.1 bool conjugate_gradient (**sparse_matrix** ∗ ***A*, matrix** ∗ ***x*, matrix** ∗ ***b***)

Solves Ax = b, x should be set to 0 Is not used, due to not working with MATLAB gate.

4.5.1.2 **sparse_matrix**∗ copy_sparse_matrix (**sparse_matrix** ∗ ***Ain***)

Copies a sparse matrix and returns it

4.5.1.3 **sparse_matrix**∗ create_empty_sparse_matrix (size_t ***size***)

Creates an empty sparse matrix

4.5.1.4 **sparse_matrix**∗ create_sparse_matrix (**matrix** ∗ ***Ain*,** int ***size***)

Creates a sparse matrix out of a normal matrix

4.5.1.5 void free_sparse_matrix (**sparse_matrix** ∗ ***S***)

Frees allocated memory of the sparse matrix

4.5.1.6 size_t matrix_sparsity (**matrix** ∗ ***A***)

Returns number of elements != 0

4.5.1.7 **matrix**∗ multiply_sparse_matrix_matrix (**sparse_matrix** ∗ ***A*, matrix** ∗ ***B***)

Multiplies sparse matrix with a normal matrix. Returns a normal matrix.

4.5.1.8 bool multiply_sparse_matrix_vector (**sparse_matrix** ∗ ***A*, matrix** ∗ ***x*, matrix** ∗ ***Ax***)

Multiplies sparse matrix with normal vector, stores result in normal matrix Ax

4.5.1.9 void print_sparse_matrix (**sparse_matrix** ∗ *S*)

Prints sparse matrix

4.5.1.10 **matrix**∗ sparse_to_normal (**sparse_matrix** ∗ *S*)

Converts sparse matrix to normal matrix

4.5.1.11 void transpose_sparse_matrix (**sparse_matrix** ∗ *Ain*)

Transposes input sparse matrix

4.5.1.12 **sparse_matrix**∗ transpose_sparse_matrix_with_return (**sparse_matrix** ∗ *Ain*)

Transposes a sparse matrix and returns it in a new sparse matrix

4.6 TDDD77/quadopt/include/problem.h File Reference

```
#include <matLib.h>
#include <work_set.h>
#include <sparse_lib.h>
```

Data Structures

- struct problem

Typedefs

- typedef struct problem problem

Functions

- problem ∗ create_problem (matrix ∗Q, matrix ∗q, matrix ∗E, matrix ∗h, matrix ∗F, matrix ∗g, matrix ∗z0, int max_iter, int max_micro_sec)
- void print_problem (problem ∗prob)
- void free_problem (problem ∗prob)
- matrix ∗ get_active_conditions (problem ∗prob)
- sparse_matrix ∗ get_sparse_active_conditions (problem ∗prob)
- matrix ∗ get_active_conditions_rhs (problem ∗prob)
- bool get_solution_value (problem ∗prob)
- void print_solution (problem ∗prob)
- bool time_to_exit (problem ∗prob, double time_spent)
- bool is_feasible_point (matrix ∗z, problem ∗prob)

4.6.1 Typedef Documentation

4.6.1.1 typedef struct **problem problem**

4.6.2 Function Documentation

4.6.2.1 **problem**∗ create_problem (**matrix** ∗ *Q,* **matrix** ∗ *q,* **matrix** ∗ *E,* **matrix** ∗ *h,* **matrix** ∗ *F,* **matrix** ∗ *g,* **matrix** ∗ *z0,* int *max_iter,* int *max_micro_sec*)

Puts matrices to a problem struct

4.6.2.2 void free_problem (**problem** ∗ *prob*)

Deallocates all the problems resources

4.6.2.3 **matrix**∗ get_active_conditions (**problem** ∗ *prob*)

Returns a matrix with the currently active constraints

4.6.2.4 **matrix**∗ get_active_conditions_rhs (**problem** ∗ *prob*)

Returns a matrix with the right hand side of the currently active constraints

4.6.2.5 bool get_solution_value (**problem** ∗ *prob*)

Calculates the optimum value given by the solution point

4.6.2.6 **sparse_matrix**∗ get_sparse_active_conditions (**problem** ∗ *prob*)

4.6.2.7 bool is_feasible_point (**matrix** ∗ *z,* **problem** ∗ *prob*)

4.6.2.8 void print_problem (**problem** ∗ *prob*)

Prints the matrices defined in the problem struct

4.6.2.9 void print_solution (**problem** ∗ *prob*)

Prints optimal point and optimal value

4.6.2.10 bool time_to_exit (**problem** ∗ *prob,* double *time_spent*)

Exits solver if maximal iterations or microseconds have been fullfilled

4.7 TDDD77/quadopt/include/simplex.h File Reference

```
#include <problem.h>
```

Functions

- bool simplex_phase_1 (problem ∗prob)

4.7.1 Function Documentation

4.7.1.1 bool simplex_phase_1 (**problem** ∗ *prob*)

4.8 TDDD77/quadopt/include/solver.h File Reference

```
#include <problem.h>
#include <matLib.h>
```

Functions

- bool remove_constraint (problem ∗prob)
- matrix ∗ quadopt_solver (problem ∗prob)

4.8.1 Function Documentation

4.8.1.1 **matrix**∗ quadopt_solver (**problem** ∗ *prob*)

Solves a quadratic problem using the active set method

4.8.1.2 bool remove_constraint (**problem** ∗ *prob*)

Removes the active constraint with the most negative lagrange multiplier

4.9 TDDD77/quadopt/include/subproblem.h File Reference

```
#include <problem.h>
```

Functions

- void solve_subproblem (problem ∗prob)

4.9.1 Function Documentation

4.9.1.1 void solve_subproblem (**problem** ∗ *prob*)

Solves the subproblem for active set

4.10 TDDD77/quadopt/include/trans_con.h File Reference

```
#include <matLib.h>
#include <stdbool.h>
```

Functions

- bool trans_dyn_cons (matrix ∗A, matrix ∗B, matrix ∗k, matrix ∗E, matrix ∗h, size_t card_x)
- bool trans_ineq_cons (matrix ∗Fx, matrix ∗gx, matrix ∗F, matrix ∗g, size_t card_x, size_t card_u, size_t N, matrix ∗x_lim, matrix ∗u_lim)
- bool create_objective (int n, matrix ∗Qin, matrix ∗P, matrix ∗R, matrix ∗Q)

4.10.1 Function Documentation

4.10.1.1 bool create_objective (int *n*, matrix ∗ *Qin*, matrix ∗ *P*, matrix ∗ *R*, matrix ∗ *Q*)

4.10.1.2 bool trans_dyn_cons (matrix ∗ *A*, matrix ∗ *B*, matrix ∗ *k*, matrix ∗ *E*, matrix ∗ *h*, size_t *card_x*)

Dynamic constraints (A and B with initial values K) transforms to equality constraints (E and h).

4.10.1.3 bool trans_ineq_cons (matrix ∗ *Fx*, matrix ∗ *gx*, matrix ∗ *F*, matrix ∗ *g*, size_t *card_x*, size_t *card_u*, size_t *N*, matrix ∗ *x_lim*, matrix ∗ *u_lim*)

4.11 TDDD77/quadopt/include/work_set.h File Reference

```
#include <stdbool.h>
```

Data Structures

- struct work_set

Typedefs

- typedef struct work_set work_set

Functions

- work_set ∗ work_set_create (size_t ws_max)
- bool work_set_free (work_set ∗ws)
- bool work_set_append (work_set ∗ws, size_t val)
- bool work_set_remove (work_set ∗ws, size_t val)
- void work_set_print (work_set ∗ws)
- bool work_set_contains (work_set ∗ws, size_t item)
- void work_set_clear (work_set ∗ws)

4.11.1 Typedef Documentation

4.11.1.1 typedef struct work_set work_set

4.11.2 Function Documentation

4.11.2.1 bool work_set_append (work_set ∗ *ws*, size_t *val*)

Adds an element to the set

4.11.2.2 void work_set_clear (**work_set** ∗ *ws*)

Clears the set

4.11.2.3 bool work_set_contains (**work_set** ∗ *ws,* size_t *item*)

Checks if the set is containing the item

4.11.2.4 **work_set**∗ work_set_create (size_t *ws_max*)

Creates a new work set

4.11.2.5 bool work_set_free (**work_set** ∗ *ws*)

Removes and deallocates the set

4.11.2.6 void work_set_print (**work_set** ∗ *ws*)

Prints all current elements in the set

4.11.2.7 bool work_set_remove (**work_set** ∗ *ws,* size_t *val*)

Removes an element from the set

4.12 TDDD77/quadopt/src/problem.c File Reference

```
#include <problem.h>
```

Functions

- void fill_constraint_matrices (problem ∗prob)
- problem ∗ create_problem (matrix ∗Q, matrix ∗q, matrix ∗E, matrix ∗h, matrix ∗F, matrix ∗g, matrix ∗z0, int max_iter, int max_micro_sec)
- void print_problem (problem ∗prob)
- void free_problem (problem ∗prob)
- matrix ∗ get_active_conditions (problem ∗prob)
- sparse_matrix ∗ get_sparse_active_conditions (problem ∗prob)
- matrix ∗ get_active_conditions_rhs (problem ∗prob)
- bool get_solution_value (problem ∗prob)
- void print_solution (problem ∗prob)
- bool time_to_exit (problem ∗prob, double time_spent)
- bool is_feasible_point (matrix ∗z, problem ∗prob)

4.12.1 Function Documentation

4.12.1.1 **problem**∗ create_problem (**matrix** ∗ *Q,* **matrix** ∗ *q,* **matrix** ∗ *E,* **matrix** ∗ *h,* **matrix** ∗ *F,* **matrix** ∗ *g,* **matrix** ∗ *z0,* int *max_iter,* int *max_micro_sec*)

Puts matrices to a problem struct

4.12.1.2 void fill_constraint_matrices (**problem** ∗ *prob*)

4.12.1.3 void free_problem (**problem** ∗ *prob*)

Deallocates all the problems resources

4.12.1.4 **matrix**∗ get_active_conditions (**problem** ∗ *prob*)

Returns a matrix with the currently active constraints

4.12.1.5 **matrix**∗ get_active_conditions_rhs (**problem** ∗ *prob*)

Returns a matrix with the right hand side of the currently active constraints

4.12.1.6 bool get_solution_value (**problem** ∗ *prob*)

Calculates the optimum value given by the solution point

4.12.1.7 **sparse_matrix**∗ get_sparse_active_conditions (**problem** ∗ *prob*)

4.12.1.8 bool is_feasible_point (**matrix** ∗ *z*, **problem** ∗ *prob*)

4.12.1.9 void print_problem (**problem** ∗ *prob*)

Prints the matrices defined in the problem struct

4.12.1.10 void print_solution (**problem** ∗ *prob*)

Prints optimal point and optimal value

4.12.1.11 bool time_to_exit (**problem** ∗ *prob*, double *time_spent*)

Exits solver if maximal iterations or microseconds have been fullfilled

4.13 TDDD77/quadopt/src/simplex.c File Reference

```
#include <simplex.h>
```

Functions

- bool is_neg_tableau_row (int row, matrix ∗tableau)
- int min_test (int column, matrix ∗tableau)
- void neg_equality (problem ∗prob, work_set ∗virtual_vars)
- void convert_geq_to_leq (problem ∗prob, work_set ∗virtual_vars, matrix ∗∗Fr, matrix ∗∗gr)
- matrix ∗ split_ineq_variables (problem ∗prob, matrix ∗Fr)
- matrix ∗ split_eq_variables (problem ∗prob)
- work_set ∗ create_basis (problem ∗prob)
- void insert_constraints (problem ∗prob, matrix ∗tableau, matrix ∗Et, matrix ∗Ft, matrix ∗gr)
- void insert_simplex_variables (problem ∗prob, work_set ∗virtual_vars, matrix ∗tableau)

- void insert_objective_function (problem ∗prob, matrix ∗tableau)
- void remove_variables (problem ∗prob, matrix ∗tableau)
- bool simplex_min (problem ∗prob, matrix ∗tableau, work_set ∗basis)
- void set_variables (problem ∗prob, work_set ∗basis, matrix ∗tableau)
- bool simplex_phase_1 (problem ∗prob)

4.13.1 Function Documentation

4.13.1.1 void convert_geq_to_leq (**problem** ∗ *prob,* **work_set** ∗ *virtual_vars,* **matrix** ∗∗ *Fr,* **matrix** ∗∗ *gr*)

4.13.1.2 **work_set**∗ create_basis (**problem** ∗ *prob*)

4.13.1.3 void insert_constraints (**problem** ∗ *prob,* **matrix** ∗ *tableau,* **matrix** ∗ *Et,* **matrix** ∗ *Ft,* **matrix** ∗ *gr*)

4.13.1.4 void insert_objective_function (**problem** ∗ *prob,* **matrix** ∗ *tableau*)

4.13.1.5 void insert_simplex_variables (**problem** ∗ *prob,* **work_set** ∗ *virtual_vars,* **matrix** ∗ *tableau*)

4.13.1.6 bool is_neg_tableau_row (int *row,* **matrix** ∗ *tableau*)

4.13.1.7 int min_test (int *column,* **matrix** ∗ *tableau*)

4.13.1.8 void neg_equality (**problem** ∗ *prob,* **work_set** ∗ *virtual_vars*)

4.13.1.9 void remove_variables (**problem** ∗ *prob,* **matrix** ∗ *tableau*)

4.13.1.10 void set_variables (**problem** ∗ *prob,* **work_set** ∗ *basis,* **matrix** ∗ *tableau*)

4.13.1.11 bool simplex_min (**problem** ∗ *prob,* **matrix** ∗ *tableau,* **work_set** ∗ *basis*)

4.13.1.12 bool simplex_phase_1 (**problem** ∗ *prob*)

4.13.1.13 **matrix**∗ split_eq_variables (**problem** ∗ *prob*)

4.13.1.14 **matrix**∗ split_ineq_variables (**problem** ∗ *prob,* **matrix** ∗ *Fr*)

4.14 TDDD77/quadopt/src/solver.c File Reference

```
#include <stdio.h>
#include <solver.h>
#include <math.h>
#include <subproblem.h>
#include <time.h>
#include <simplex.h>
```

Functions

- bool fill_active_set (problem ∗prob)
- bool take_step (problem ∗prob)
- void copy_solution (problem ∗prob)
- void prefill_set (problem ∗prob)
- bool remove_constraint (problem ∗prob)
- matrix ∗ quadopt_solver (problem ∗prob)

4.14.1 Function Documentation

4.14.1.1 void copy_solution (**problem** ∗ *prob*)

4.14.1.2 bool fill_active_set (**problem** ∗ *prob*)

4.14.1.3 void prefill_set (**problem** ∗ *prob*)

4.14.1.4 **matrix**∗ quadopt_solver (**problem** ∗ *prob*)

Solves a quadratic problem using the active set method

4.14.1.5 bool remove_constraint (**problem** ∗ *prob*)

Removes the active constraint with the most negative lagrange multiplier

4.14.1.6 bool take_step (**problem** ∗ *prob*)

4.15 TDDD77/quadopt/src/subproblem.c File Reference

```
#include <subproblem.h>
#include <matLib.h>
#include <solver.h>
#include <assert.h>
```

Functions

- void range_space_sparse (sparse_matrix ∗A, problem ∗prob)
- void range_space (matrix ∗A, problem ∗prob)
- void KKT_sub_sparse (sparse_matrix ∗A, problem ∗prob)
- void KKT_sub (matrix ∗A, problem ∗prob)
- void solve_subproblem (problem ∗prob)

4.15.1 Function Documentation

4.15.1.1 void KKT_sub (**matrix** ∗ *A,* **problem** ∗ *prob*)

4.15.1.2 void KKT_sub_sparse (**sparse_matrix** ∗ *A,* **problem** ∗ *prob*)

4.15.1.3 void range_space (**matrix** ∗ *A,* **problem** ∗ *prob*)

4.15.1.4 void range_space_sparse (**sparse_matrix** ∗ *A,* **problem** ∗ *prob*)

4.15.1.5 void solve_subproblem (**problem** ∗ *prob*)

Solves the subproblem for active set

4.16 TDDD77/quadopt/src/trans_con.c File Reference

```
#include <trans_con.h>
#include <assert.h>
```

Functions

- bool insert_x_identity_matrices (matrix ∗F, size_t card_x, size_t N)
- bool insert_fx (matrix ∗F, matrix ∗Fx, size_t card_x, size_t N)
- bool insert_u_identity_matrices (matrix ∗F, size_t card_u, size_t N)
- bool fix_g (matrix ∗g, matrix ∗gx, matrix ∗x_lim, matrix ∗u_lim, size_t N)
- bool insert_identity_matrices (matrix ∗E, size_t card_x)
- bool insert_A_matrices (matrix ∗E, matrix ∗A)
- bool insert_B_matrices (matrix ∗E, matrix ∗B, size_t N)
- bool trans_dyn_cons (matrix ∗A, matrix ∗B, matrix ∗k, matrix ∗E, matrix ∗h, size_t card_x)
- bool trans_ineq_cons (matrix ∗Fx, matrix ∗gx, matrix ∗F, matrix ∗g, size_t card_x, size_t card_u, size_t N, matrix ∗x_lim, matrix ∗u_lim)
- bool create_objective (int n, matrix ∗Qin, matrix ∗P, matrix ∗R, matrix ∗Q)

4.16.1 Function Documentation

4.16.1.1 bool create_objective (int *n*, matrix ∗ *Qin*, matrix ∗ *P*, matrix ∗ *R*, matrix ∗ *Q*)

4.16.1.2 bool fix_g (matrix ∗ *g*, matrix ∗ *gx*, matrix ∗ *x_lim*, matrix ∗ *u_lim*, size_t *N*)

4.16.1.3 bool insert_A_matrices (matrix ∗ *E*, matrix ∗ *A*)

4.16.1.4 bool insert_B_matrices (matrix ∗ *E*, matrix ∗ *B*, size_t *N*)

4.16.1.5 bool insert_fx (matrix ∗ *F*, matrix ∗ *Fx*, size_t *card_x*, size_t *N*)

4.16.1.6 bool insert_identity_matrices (matrix ∗ *E*, size_t *card_x*)

4.16.1.7 bool insert_u_identity_matrices (matrix ∗ *F*, size_t *card_u*, size_t *N*)

4.16.1.8 bool insert_x_identity_matrices (matrix ∗ *F*, size_t *card_x*, size_t *N*)

4.16.1.9 bool trans_dyn_cons (matrix ∗ *A*, matrix ∗ *B*, matrix ∗ *k*, matrix ∗ *E*, matrix ∗ *h*, size_t *card_x*)

Dynamic constraints (A and B with initial values K) transforms to equality constraints (E and h).

4.16.1.10 bool trans_ineq_cons (matrix ∗ *Fx*, matrix ∗ *gx*, matrix ∗ *F*, matrix ∗ *g*, size_t *card_x*, size_t *card_u*, size_t *N*, matrix ∗ *x_lim*, matrix ∗ *u_lim*)

4.17 TDDD77/quadopt/src/work_set.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <work_set.h>
```

Functions

- work_set ∗ work_set_create (size_t ws_max)
- bool work_set_append (work_set ∗ws, size_t val)
- bool work_set_remove (work_set ∗ws, size_t val)

- bool work_set_free (work_set ∗ws)
- void work_set_print (work_set ∗ws)
- bool work_set_contains (work_set ∗ws, size_t item)
- void work_set_clear (work_set ∗ws)

4.17.1 Function Documentation

4.17.1.1 bool work_set_append (work_set ∗ *ws,* size_t *val*)

Adds an element to the set

4.17.1.2 void work_set_clear (work_set ∗ *ws*)

Clears the set

4.17.1.3 bool work_set_contains (work_set ∗ *ws,* size_t *item*)

Checks if the set is containing the item

4.17.1.4 work_set∗ work_set_create (size_t *ws_max*)

Creates a new work set

4.17.1.5 bool work_set_free (work_set ∗ *ws*)

Removes and deallocates the set

4.17.1.6 void work_set_print (work_set ∗ *ws*)

Prints all current elements in the set

4.17.1.7 bool work_set_remove (work_set ∗ *ws,* size_t *val*)

Removes an element from the set