QuadOpt

v1.0

Generated by Doxygen 1.8.9.1

Thu Apr 23 2015 16:12:29

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 matrices Struct Reference

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

Data Fields

- matrix * one
- matrix * two
- matrix * three
- matrix * four
- matrix * five

3.1.1 Field Documentation

3.1.1.1 matrix* five

3.1.1.2 matrix* four

3.1.1.3 matrix* one

3.1.1.4 matrix* three

3.1.1.5 matrix* two

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

- TDDD77/matrixlibrary/include/matLib.h

3.2 matrix Struct Reference

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

Data Fields

- int columns
- int rows

- size_t size
- value * start
- bool diagonals

3.2.1 Detailed Description

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

3.2.2 Field Documentation

3.2.2.1 int columns

3.2.2.2 bool diagonals

3.2.2.3 int rows

3.2.2.4 size_t size

3.2.2.5 value* start

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

- TDDD77/matrixlibrary/include/matLib.h

3.3 problem Struct Reference

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

Data Fields

- matrix * Q
- matrix * Q_inv
- matrix * q
- int equality_count
- matrix * E
- matrix * h
- int inequality_count
- matrix * F
- matrix * g
- matrix * A
- matrix * b
- int 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

3.3.1 Detailed Description

Allocates the problem and sets all necessary variables

3.3.2 Field Documentation

3.3.2.1 matrix* A

All constraints left-hand side coefficients.

3.3.2.2 value accuracy

3.3.2.3 work_set* active_set

The active constraints.

3.3.2.4 matrix* b

All constraints right-hand side constraints.

3.3.2.5 bool check_time

3.3.2.6 int constraints_count

Total number of constraints.

3.3.2.7 matrix* E

Equality constraints left-hand side coefficient.

3.3.2.8 int equality_count

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

3.3.2.9 matrix* F

Larger-than constraints left-hand side coefficient.

3.3.2.10 matrix* g

Larger-than constraints right-hand side constraint.

3.3.2.11 matrix∗ gk

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

See also

Q
z
q

3.3.2.12 matrix∗ h

Equality constraints right-hand side constraint.

3.3.2.13 bool has_solution

3.3.2.14 bool has_start_point

3.3.2.15 int inequality_count

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

3.3.2.16 matrix∗ lagrange

The lagrange multipliers.

3.3.2.17 int max_iter

3.3.2.18 int max_micro_sec

3.3.2.19 matrix∗ p

Current step direction towards the solution.

3.3.2.20 matrix∗ Q

The matrix containing the quadratic optimization problem.

3.3.2.21 matrix∗ q

The matrix containing the linear optimization problem.

3.3.2.22 matrix∗ Q_inv

Q inverse.

3.3.2.23 matrix∗ solution

The final point in the solution.

3.3.2.24 value solution_value

The value of the solution point.

3.3.2.25 value step

How far we will step towards the solution.

3.3.2.26 matrix* z

The current point in the solution.

3.3.2.27 matrix* z0

The starting point for the solution.

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

- TDDD77/quadopt/include/problem.h

3.4 work_set Struct Reference

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

Data Fields

- int count
- int * data

3.4.1 Detailed Description

Structure for storing different sets

3.4.2 Field Documentation

3.4.2.1 int count

Number of elements in the work set.

3.4.2.2 int* data

Array 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[ ])

Variables

- mxArray ∗ mat_matrix
- double ∗ out_matrix
- matrix ∗ lib_matrix
- matrix ∗ result_matrix
- matrix ∗ lib_matrices [nrhs-1]

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.

4.1.2 Variable Documentation

4.1.2.1 matrix∗ lib_matrices[nrhs-1]

An array of all the matrices that should be sent to the solver.

4.1.2.2 matrix∗ lib_matrix

Used to temporarily store the Matlib matrix that is created when converting the Matlab matrix.

4.1.2.3 mxArray* mat_matrix

Used to store the incoming matrices from Matlab when converting to Matlib matrices.

4.1.2.4 double* out_matrix

Used to return the result back to Matlab.

4.1.2.5 matrix* result_matrix

The Matlib matrix containing the result returned from the solver.

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

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

Data Structures

- struct matrix
- struct matrices

Macros

- #define FLOAT
- #define FORMAT_STRING "%f "
- #define PRECISION 0.01

Typedefs

- typedef float value
- typedef struct matrix matrix
- typedef struct matrices matrices

Functions

- matrix * create_matrix (int row, int col)
- matrix * create_zero_matrix (int row, int col)
- matrix * create_identity_matrix (int row, int col)
- value dot_product (matrix *r, matrix *v)
- void free_matrix (matrix *mat)
- void print_matrix (matrix *mat)
- bool check_boundaries (int row, int 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, int row, int col, matrix *mat)

- void insert_value_without_check (value insert, int row, int col, matrix *mat)
- value get_value (int row, int col, matrix *mat)
- value get_value_without_check (int row, int 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_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 * strassen_matrices_parallel_with_return (matrix *a, matrix *b)
- void * calculation_one (void *arg)
- void * calculation_two (void *arg)
- void * calculation_three (void *arg)
- void * calculation_four (void *arg)
- void * calculation_five (void *arg)
- void * calculation_six (void *arg)
- void * calculation_seven (void *arg)
- bool strassen_matrices_parallel (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)
- 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)
- matrix * get_matrix_with_only_pivots (matrix *a)
- value min (value a, value b)
- int largest_element_in_column_index (int column, int start, matrix *a)
- int smallest_element_in_column_index (int column, int start, matrix *a)
- int first_nonezero_in_column_index (int column, int start, matrix *a)
- int first_nonezero_in_row_index (int row, int start, matrix *a)
- void add_rows (int row1, int row2, matrix *a)
- bool transpose_matrix (matrix *a, matrix *b)
- matrix * transpose_matrix_with_return (matrix *a)
- value sum_of_row (int row, matrix *mat)
- value sum_of_column (int column, matrix *mat)
- value product_of_row (int row, matrix *mat)
- value product_of_column (int 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, int row, matrix *mat)
- void divide_row_with_scalar (value scal, int row, matrix *mat)
- void multiply_column_with_scalar (value scal, int col, matrix *mat)
- void divide_column_with_scalar (value scal, int col, matrix *mat)
- bool get_row_vector (int row, matrix *a, matrix *b)
- matrix * get_row_vector_with_return (int row, matrix *a)
- bool insert_row_vector (int row, matrix *a, matrix *b)
- bool switch_rows (int row1, int row2, matrix *a)
- bool get_column_vector (int column, matrix *a, matrix *b)
- matrix * get_column_vector_with_return (int column, matrix *a)
- bool insert_column_vector (int column, matrix *a, matrix *b)

- bool get_sub_matrix (int start_row, int end_row, int start_col, int end_col, matrix ∗a, matrix ∗b)
- bool insert_sub_matrix (int start_row, int end_row, int start_col, int 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 (int 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 (int rows, int columns)

4.2.1 Macro Definition Documentation

4.2.1.1 #define FLOAT

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 0.01

4.2.2 Typedef Documentation

4.2.2.1 typedef struct matrices matrices

4.2.2.2 typedef struct matrix matrix

4.2.2.3 typedef float 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 (int *row1*, int *row2*, matrix ∗ *a*)

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

4.2.3.4 void* calculation_five (void * *arg*)

4.2.3.5 void* calculation_four (void * *arg*)

4.2.3.6 void* calculation_one (void * *arg*)

4.2.3.7 void* calculation_seven (void * *arg*)

4.2.3.8 void* calculation_six (void * *arg*)

4.2.3.9 void* calculation_three (void * *arg*)

4.2.3.10 void* calculation_two (void * *arg*)

4.2.3.11 bool check_boundaries (int *row*, int *col*, matrix * *mat*)

Checks if the position exists in the matrix

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

Compare two element values

4.2.3.13 bool compare_matrices (matrix * *a*, matrix * *b*)

Returns true if matrices a and b look the same

4.2.3.14 matrix* create_identity_matrix (int *row*, int *col*)

Creates a identity matrix

4.2.3.15 matrix* create_matrix (int *row*, int *col*)

Create a matrix

4.2.3.16 matrix* create_zero_matrix (int *row*, int *col*)

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

4.2.3.17 bool crout (matrix * *a*, matrix * *l*, matrix * *u*)

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

4.2.3.18 matrix* derivate_matrix_with_return (int *var*, matrix * *a*)

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

4.2.3.19 void divide_column_with_scalar (value *scal*, int *col*, matrix * *mat*)

Divides a column with a scalar

4.2.3.20 void divide_matrix_with_scalar (value *scal*, matrix * *mat*)

Divides matrix mat with scalar

4.2.3.21 void divide_row_with_scalar (value *scal*, int *row*, matrix * *mat*)

Divides a row with a scalar

4.2.3.22 value dot_product (matrix * *r*, matrix * *v*)

Calculate the dot product

4.2.3.23 int first_nonezero_in_column_index (int *column*, int *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

4.2.3.24 int first_nonezero_in_row_index (int *row*, int *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

4.2.3.25 void forward_backward (matrix * *l*, matrix * *u*, matrix * *x*, matrix * *b*)

Solves lux=b using backward and forward substitution

4.2.3.26 void free_matrix (matrix * *mat*)

Destroy a matrix

4.2.3.27 bool gauss_jordan (matrix * *a*)

Gauss eliminates the matrix a

4.2.3.28 bool get_column_vector (int *column*, matrix * *a*, matrix * *b*)

Takes column vector from matrix a and puts it into b

4.2.3.29 matrix* get_column_vector_with_return (int *column*, matrix * *a*)

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

4.2.3.30 value get_determinant (matrix * *a*)

Returns the determinant of matrix a

4.2.3.31 bool get_diagonal (matrix * *a*, matrix * *b*)

Takes the diagonal in a and puts it into b

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

Calculates the inverse of a and puts it into c

4.2.3.33 **matrix**∗ get_matrix_with_only_pivots (**matrix** ∗ *a*)

Returns a matrix with only pivots elements from a

4.2.3.34 bool get_row_vector (int *row,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Takes row vector from matrix a and puts it into b

4.2.3.35 **matrix**∗ get_row_vector_with_return (int *row,* **matrix** ∗ *a*)

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

4.2.3.36 bool get_sub_matrix (int *start_row,* int *end_row,* int *start_col,* int *end_col,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Get a sub matrix from a

4.2.3.37 **value** get_value (int *row,* int *col,* **matrix** ∗ *mat*)

Get a value from matrix

4.2.3.38 **value** get_value_without_check (int *row,* int *col,* **matrix** ∗ *mat*)

As get_value without check

4.2.3.39 **matrix**∗ get_zero_matrix (int *rows,* int *columns*)

Creates new matrix with zero values

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

Insert a array into the matrix

4.2.3.41 bool insert_column_vector (int *column,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Inserts column vector a into matrix b at position column

4.2.3.42 bool insert_row_vector (int *row,* **matrix** ∗ *a,* **matrix** ∗ *b*)

Inserts row vector a into b:s row

4.2.3.43 bool insert_sub_matrix (int *start_row,* int *end_row,* int *start_col,* int *end_col,* **matrix** ∗ *b,* **matrix** ∗ *a*)

4.2.3.44 bool insert_value (**value** *insert,* int *row,* int *col,* **matrix** ∗ *mat*)

Insert a value into matrix

4.2.3.45 void insert_value_without_check (value *insert*, int *row*, int *col*, matrix ∗ *mat*)

As insert_value without check

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

Return true if the matrix are the same

4.2.3.47 bool is_non_negative_diagonal_matrix (matrix ∗ *A*)

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

4.2.3.48 bool is_non_negative_matrix (matrix ∗ *v*)

Checks if all elements in a matrix is positive

4.2.3.49 bool is_zero_matrix (matrix ∗ *v*)

Checks if all elements in a matrix is equal to zero

4.2.3.50 int largest_element_in_column_index (int *column*, int *start*, matrix ∗ *a*)

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

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

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

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

Return true if b contains value a

4.2.3.53 matrix∗ matrix_copy (matrix ∗ *source*)

Copy and return new matrix.

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

Copies all the data from matrix A into matrix B

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

Returns the lowest of the two values

4.2.3.56 void multiply_column_with_scalar (value *scal*, int *col*, matrix ∗ *mat*)

Multiplies a column with a scalar

4.2.3.57 bool multiply_matrices (matrix * *a*, matrix * *b*, matrix * *c*)

Multiply a and b into c. c=a*b

4.2.3.58 bool multiply_matrices_optimized (matrix * *a*, matrix * *b*, matrix * *c*)

4.2.3.59 matrix* multiply_matrices_with_return (matrix * *a*, matrix * *b*)

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

4.2.3.60 void multiply_matrix_with_scalar (value *scal*, matrix * *mat*)

Multiplies matrix mat with scalar

4.2.3.61 void multiply_row_with_scalar (value *scal*, int *row*, matrix * *mat*)

Multiplies a row with a scalar

4.2.3.62 void print_matrix (matrix * *mat*)

Prints the matrix

4.2.3.63 value product_of_column (int *column*, matrix * *mat*)

Return the product of a column in matrix mat

4.2.3.64 value product_of_row (int *row*, matrix * *mat*)

Return the product of a row in matrix mat

4.2.3.65 int smallest_element_in_column_index (int *column*, int *start*, matrix * *a*)

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

4.2.3.66 bool solve_linear (matrix * *a*, matrix * *x*, matrix * *b*)

Solves Ax=B

4.2.3.67 matrix* solve_linear_with_return (matrix * *a*, matrix * *b*)

Solves ax=b by returning a pointer to x

4.2.3.68 bool strassen_matrices (matrix * *a*, matrix * *b*, matrix * *c*)

4.2.3.69 bool strassen_matrices_parallel (matrix * *a*, matrix * *b*, matrix * *c*)

4.2.3.70 matrix* strassen_matrices_parallel_with_return (matrix * *a*, matrix * *b*)

4.2.3.71 matrix* strassen_matrices_with_return (matrix * *a*, matrix * *b*)

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

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

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

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

4.2.3.74 **value** sum_of_column (int *column*, **matrix** ∗ *mat*)

Return the sum of a column in matrix mat

4.2.3.75 **value** sum_of_row (int *row*, **matrix** ∗ *mat*)

Return the sum of a row in matrix mat

4.2.3.76 bool switch_rows (int *row1*, int *row2*, **matrix** ∗ *a*)

Switches rows in a

4.2.3.77 void transform_to_reduced_row_echelon_form (**matrix** ∗ *M*)

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

Transposes matrix a into b

4.2.3.79 **matrix**∗ transpose_matrix_with_return (**matrix** ∗ *a*)

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

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

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

Functions

- matrix ∗ create_matrix (int row, int col)
- matrix ∗ create_zero_matrix (int row, int col)
- matrix ∗ create_identity_matrix (int row, int col)
- void free_matrix (matrix ∗mat)
- value dot_product (matrix ∗r, matrix ∗v)
- void print_matrix (matrix ∗mat)
- bool check_boundaries (int row, int 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, int row, int col, matrix ∗mat)
- void insert_value_without_check (value insert, int row, int col, matrix ∗mat)

- value get_value (int row, int col, matrix *mat)
- value get_value_without_check (int row, int 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_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 * strassen_matrices_parallel_with_return (matrix *a, matrix *b)
- void * calculation_one (void *arg)
- void * calculation_two (void *arg)
- void * calculation_three (void *arg)
- void * calculation_four (void *arg)
- void * calculation_five (void *arg)
- void * calculation_six (void *arg)
- void * calculation_seven (void *arg)
- bool strassen_matrices_parallel (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)
- 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)
- matrix * get_matrix_with_only_pivots (matrix *a)
- value min (value a, value b)
- int largest_element_in_column_index (int column, int start, matrix *a)
- int smallest_element_in_column_index (int column, int start, matrix *a)
- int first_nonezero_in_column_index (int column, int start, matrix *a)
- int first_nonezero_in_row_index (int row, int start, matrix *a)
- void add_rows (int row1, int row2, matrix *a)
- bool transpose_matrix (matrix *a, matrix *b)
- matrix * transpose_matrix_with_return (matrix *a)
- value sum_of_row (int row, matrix *mat)
- value sum_of_column (int column, matrix *mat)
- value product_of_row (int row, matrix *mat)
- value product_of_column (int 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, int row, matrix *mat)
- void divide_row_with_scalar (value scal, int row, matrix *mat)
- void multiply_column_with_scalar (value scal, int col, matrix *mat)
- void divide_column_with_scalar (value scal, int col, matrix *mat)
- bool get_row_vector (int row, matrix *a, matrix *b)
- matrix * get_row_vector_with_return (int row, matrix *a)
- bool insert_row_vector (int row, matrix *a, matrix *b)
- bool switch_rows (int row1, int row2, matrix *a)
- bool get_column_vector (int column, matrix *a, matrix *b)
- matrix * get_column_vector_with_return (int column, matrix *a)
- bool insert_column_vector (int column, matrix *a, matrix *b)
- bool get_sub_matrix (int start_row, int end_row, int start_col, int end_col, matrix *a, matrix *b)

- bool insert_sub_matrix (int start_row, int end_row, int start_col, int 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 (int 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 (int rows, int columns)

4.3.1 Function Documentation

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

Adds a and b into c

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

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

4.3.1.3 void add_rows (int *row1*, int *row2*, **matrix** ∗ *a*)

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

4.3.1.4 void∗ calculation_five (void ∗ *arg*)

4.3.1.5 void∗ calculation_four (void ∗ *arg*)

4.3.1.6 void∗ calculation_one (void ∗ *arg*)

4.3.1.7 void∗ calculation_seven (void ∗ *arg*)

4.3.1.8 void∗ calculation_six (void ∗ *arg*)

4.3.1.9 void∗ calculation_three (void ∗ *arg*)

4.3.1.10 void∗ calculation_two (void ∗ *arg*)

4.3.1.11 bool check_boundaries (int *row*, int *col*, **matrix** ∗ *mat*)

Checks if the position exists in the matrix

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

Compare two element values

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

Returns true if matrices a and b look the same

4.3.1.14 matrix* create_identity_matrix (int *row*, int *col*)

Creates a identity matrix

4.3.1.15 matrix* create_matrix (int *row*, int *col*)

Create a matrix

4.3.1.16 matrix* create_zero_matrix (int *row*, int *col*)

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

4.3.1.17 bool crout (matrix * *a*, matrix * *l*, matrix * *u*)

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

4.3.1.18 matrix* derivate_matrix_with_return (int *var*, matrix * *a*)

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

4.3.1.19 void divide_column_with_scalar (value *scal*, int *col*, matrix * *mat*)

Divides a column with a scalar

4.3.1.20 void divide_matrix_with_scalar (value *scal*, matrix * *mat*)

Divides matrix mat with scalar

4.3.1.21 void divide_row_with_scalar (value *scal*, int *row*, matrix * *mat*)

Divides a row with a scalar

4.3.1.22 value dot_product (matrix * *r*, matrix * *v*)

Calculate the dot product

4.3.1.23 int first_nonezero_in_column_index (int *column*, int *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

4.3.1.24 int first_nonezero_in_row_index (int *row*, int *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

4.3.1.25 void forward_backward (matrix * *l*, matrix * *u*, matrix * *x*, matrix * *b*)

Solves lux=b using backward and forward substitution

4.3.1.26 void free_matrix (**matrix** * *mat*)

Destroy a matrix

4.3.1.27 bool gauss_jordan (**matrix** * *a*)

Gauss eliminates the matrix a

4.3.1.28 bool get_column_vector (int *column*, **matrix** * *a*, **matrix** * *b*)

Takes column vector from matrix a and puts it into b

4.3.1.29 **matrix*** get_column_vector_with_return (int *column*, **matrix** * *a*)

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

4.3.1.30 **value** get_determinant (**matrix** * *a*)

Returns the determinant of matrix a

4.3.1.31 bool get_diagonal (**matrix** * *a*, **matrix** * *b*)

Takes the diagonal in a and puts it into b

4.3.1.32 bool get_inverse (**matrix** * *a*, **matrix** * *c*)

Calculates the inverse of a and puts it into c

4.3.1.33 **matrix*** get_matrix_with_only_pivots (**matrix** * *a*)

Returns a matrix with only pivots elements from a

4.3.1.34 bool get_row_vector (int *row*, **matrix** * *a*, **matrix** * *b*)

Takes row vector from matrix a and puts it into b

4.3.1.35 **matrix*** get_row_vector_with_return (int *row*, **matrix** * *a*)

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

4.3.1.36 bool get_sub_matrix (int *start_row*, int *end_row*, int *start_col*, int *end_col*, **matrix** * *a*, **matrix** * *b*)

Get a sub matrix from a

4.3.1.37 **value** get_value (int *row*, int *col*, **matrix** * *mat*)

Get a value from matrix

4.3.1.38 value get_value_without_check (int *row*, int *col*, matrix * *mat*)

As get_value without check

4.3.1.39 matrix* get_zero_matrix (int *rows*, int *columns*)

Creates new matrix with zero values

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

Insert a array into the matrix

4.3.1.41 bool insert_column_vector (int *column*, matrix * *a*, matrix * *b*)

Inserts column vector a into matrix b at position column

4.3.1.42 bool insert_row_vector (int *row*, matrix * *a*, matrix * *b*)

Inserts row vector a into b:s row

4.3.1.43 bool insert_sub_matrix (int *start_row*, int *end_row*, int *start_col*, int *end_col*, matrix * *b*, matrix * *a*)

4.3.1.44 bool insert_value (value *insert*, int *row*, int *col*, matrix * *mat*)

Insert a value into matrix

4.3.1.45 void insert_value_without_check (value *insert*, int *row*, int *col*, matrix * *mat*)

As insert_value without check

4.3.1.46 bool is_matrix (matrix * *a*, matrix * *b*)

Return true if the matrix are the same

4.3.1.47 bool is_non_negative_diagonal_matrix (matrix * *A*)

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

4.3.1.48 bool is_non_negative_matrix (matrix * *v*)

Checks if all elements in a matrix is positive

4.3.1.49 bool is_zero_matrix (matrix * *v*)

Checks if all elements in a matrix is equal to zero

4.3.1.50 int largest_element_in_column_index (int *column*, int *start*, matrix * *a*)

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

4.3.1.51 void least_square (matrix * *a*, matrix * *x*, matrix * *b*)

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

4.3.1.52 bool matrix_contains (value *a*, matrix * *b*)

Return true if b contains value a

4.3.1.53 matrix* matrix_copy (matrix * *source*)

Copy and return new matrix.

4.3.1.54 void matrix_copy_data (matrix * *A*, matrix * *B*)

Copies all the data from matrix A into matrix B

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

Returns the lowest of the two values

4.3.1.56 void multiply_column_with_scalar (value *scal*, int *col*, matrix * *mat*)

Multiplies a column with a scalar

4.3.1.57 bool multiply_matrices (matrix * *a*, matrix * *b*, matrix * *c*)

Multiply a and b into c. c=a*b

4.3.1.58 bool multiply_matrices_optimized (matrix * *a*, matrix * *b*, matrix * *c*)

4.3.1.59 matrix* multiply_matrices_with_return (matrix * *a*, matrix * *b*)

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

4.3.1.60 void multiply_matrix_with_scalar (value *scal*, matrix * *mat*)

Multiplies matrix mat with scalar

4.3.1.61 void multiply_row_with_scalar (value *scal*, int *row*, matrix * *mat*)

Multiplies a row with a scalar

4.3.1.62 void print_matrix (matrix * *mat*)

Prints the matrix

4.3.1.63 value product_of_column (int *column*, matrix * *mat*)

Return the product of a column in matrix mat

4.3.1.64 value product_of_row (int *row,* matrix ∗ *mat*)

Return the product of a row in matrix mat

4.3.1.65 int smallest_element_in_column_index (int *column,* int *start,* matrix ∗ *a*)

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

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

Solves Ax=B

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

Solves ax=b by returning a pointer to x

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

4.3.1.69 bool strassen_matrices_parallel (matrix ∗ *a,* matrix ∗ *b,* matrix ∗ *c*)

4.3.1.70 matrix∗ strassen_matrices_parallel_with_return (matrix ∗ *a,* matrix ∗ *b*)

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

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

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

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

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

4.3.1.74 value sum_of_column (int *column,* matrix ∗ *mat*)

Return the sum of a column in matrix mat

4.3.1.75 value sum_of_row (int *row,* matrix ∗ *mat*)

Return the sum of a row in matrix mat

4.3.1.76 bool switch_rows (int *row1,* int *row2,* matrix ∗ *a*)

Switches rows in a

4.3.1.77 void transform_to_reduced_row_echelon_form (matrix ∗ *M*)

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

Transposes matrix a into b

4.3.1.79 matrix* transpose_matrix_with_return (matrix * *a*)

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

4.4 TDDD77/quadopt/include/feasible_point.h File Reference

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

Functions

- bool is_feasible_point (matrix *z, problem *prob)
- bool find_starting_point (problem *prob)

4.4.1 Function Documentation

4.4.1.1 bool find_starting_point (problem * *prob*)

Calculates a feasible starting point for a problem

4.4.1.2 bool is_feasible_point (matrix * *z*, problem * *prob*)

Checks if a point is feasible subject to the constraints in a problem

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

```
#include <matLib.h>
#include <work_set.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)
- 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)

4.5.1 Typedef Documentation

4.5.1.1 typedef struct problem problem

4.5.2 Function Documentation

4.5.2.1 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

4.5.2.2 void free_problem (problem ∗ *prob*)

Deallocates all the problems resources

4.5.2.3 matrix∗ get_active_conditions (problem ∗ *prob*)

Returns a matrix with the currently active constraints

4.5.2.4 matrix∗ get_active_conditions_rhs (problem ∗ *prob*)

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

4.5.2.5 bool get_solution_value (problem ∗ *prob*)

Calculates the optimum value given by the solution point

4.5.2.6 void print_problem (problem ∗ *prob*)

Prints the matrices defined in the problem struct

4.5.2.7 void print_solution (problem ∗ *prob*)

Prints optimal point and optimal value

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

Exits solver if maximal iterations or microseconds have been fullfilled

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

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

Functions

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

4.6.1 Function Documentation

4.6.1.1 matrix* quadopt_solver (problem * *prob*)

Solves a quadratic problem using the active set method

4.6.1.2 bool remove_constraint (problem * *prob*)

Removes the active constraint with the most negative lagrange multiplier

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

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

Functions

- void solve_subproblem (problem *prob)

4.7.1 Function Documentation

4.7.1.1 void solve_subproblem (problem * *prob*)

Solves the subproblem for active set

4.8 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 (int ws_max)
- bool work_set_free (work_set *ws)
- bool work_set_append (work_set *ws, int val)
- bool work_set_remove (work_set *ws, int val)
- void work_set_print (work_set *ws)
- bool work_set_contains (work_set *ws, int item)
- void work_set_clear (work_set *ws)

4.8.1 Typedef Documentation

4.8.1.1 typedef struct work_set work_set

4.8.2 Function Documentation

4.8.2.1 bool work_set_append (work_set ∗ *ws,* int *val*)

Adds an element to the set

4.8.2.2 void work_set_clear (work_set ∗ *ws*)

Clears the set

4.8.2.3 bool work_set_contains (work_set ∗ *ws,* int *item*)

Checks if the set is containing the item

4.8.2.4 work_set∗ work_set_create (int *ws_max*)

Creates a new work set

4.8.2.5 bool work_set_free (work_set ∗ *ws*)

Removes and deallocates the set

4.8.2.6 void work_set_print (work_set ∗ *ws*)

Prints all current elements in the set

4.8.2.7 bool work_set_remove (work_set ∗ *ws,* int *val*)

Removes an element from the set

4.9 TDDD77/quadopt/src/feasible_point.c File Reference

```
#include <feasible_point.h>
```

Functions

- void comb (int pool, int need, int ∗rows, int at, int ri, problem ∗prob, matrix ∗A, matrix ∗b, matrix ∗z, bool ∗done)
- bool is_feasible_point (matrix ∗z, problem ∗prob)
- bool find_starting_point (problem ∗prob)

4.9.1 Function Documentation

4.9.1.1 void comb (int *pool*, int *need*, int * *rows*, int *at*, int *ri*, problem * *prob*, matrix * *A*, matrix * *b*, matrix * *z*, bool * *done*)

4.9.1.2 bool find_starting_point (problem * *prob*)

Calculates a feasible starting point for a problem

4.9.1.3 bool is_feasible_point (matrix * *z*, problem * *prob*)

Checks if a point is feasible subject to the constraints in a problem

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

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

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)
- 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)

4.10.1 Function Documentation

4.10.1.1 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

4.10.1.2 void free_problem (problem * *prob*)

Deallocates all the problems resources

4.10.1.3 matrix* get_active_conditions (problem * *prob*)

Returns a matrix with the currently active constraints

4.10.1.4 matrix* get_active_conditions_rhs (problem * *prob*)

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

4.10.1.5 bool get_solution_value (problem ∗ *prob*)

Calculates the optimum value given by the solution point

4.10.1.6 void print_problem (problem ∗ *prob*)

Prints the matrices defined in the problem struct

4.10.1.7 void print_solution (problem ∗ *prob*)

Prints optimal point and optimal value

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

Exits solver if maximal iterations or microseconds have been fullfilled

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

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

Functions

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

4.11.1 Function Documentation

4.11.1.1 void copy_solution (problem ∗ *prob*)

4.11.1.2 bool fill_active_set (problem ∗ *prob*)

4.11.1.3 matrix∗ quadopt_solver (problem ∗ *prob*)

Solves a quadratic problem using the active set method

4.11.1.4 bool remove_constraint (problem ∗ *prob*)

Removes the active constraint with the most negative lagrange multiplier

4.11.1.5 bool take_step (problem * prob)

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

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

Functions

- void solve_subproblem (problem *prob)

4.12.1 Function Documentation

4.12.1.1 void solve_subproblem (problem * prob)

Solves the subproblem for active set

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

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

Functions

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

4.13.1 Function Documentation

4.13.1.1 bool work_set_append (work_set * ws, int val)

Adds an element to the set

4.13.1.2 void work_set_clear (work_set * ws)

Clears the set

4.13.1.3 bool work_set_contains (work_set * ws, int item)

Checks if the set is containing the item

4.13.1.4 work_set* work_set_create (int *ws_max*)

Creates a new work set

4.13.1.5 bool work_set_free (work_set * *ws*)

Removes and deallocates the set

4.13.1.6 void work_set_print (work_set * *ws*)

Prints all current elements in the set

4.13.1.7 bool work_set_remove (work_set * *ws,* int *val*)

Removes an element from the set