

Project 2 - FYS4150

Generated by Doxygen 1.14.0

1 Topic Index	1
1.1 Topics	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Topic Documentation	7
4.1 Standalone Functions	7
4.1.1 Detailed Description	7
4.1.2 Function Documentation	7
4.1.2.1 analytic_solution()	7
4.1.2.2 create_tridiagonal()	8
4.1.2.3 jacobi_eigensolver()	8
4.1.2.4 jacobi_rotate()	9
4.1.2.5 max_offdiag_symmetric()	9
5 Class Documentation	11
5.1 Args Struct Reference	11
5.1.1 Detailed Description	11
5.1.2 Member Data Documentation	12
5.1.2.1 maxiter	12
5.1.2.2 N_max	12
5.1.2.3 n_steps	12
5.1.2.4 outfile	12
5.1.2.5 run_problem_5	12
5.1.2.6 run_problem_6	12
5.1.2.7 run_tests	12
5.1.2.8 tol	13
5.2 TriDag Class Reference	13
5.2.1 Detailed Description	13
5.2.2 Constructor & Destructor Documentation	14
5.2.2.1 TriDag()	14
5.2.3 Member Function Documentation	14
5.2.3.1 compute_eigenvalues()	14
5.2.3.2 print()	14
5.2.4 Member Data Documentation	14
5.2.4.1 A	14
5.2.4.2 eigenvalues	14
5.2.4.3 eigenvectors	14
6 File Documentation	15

6.1 arg_parser.hpp File Reference	15
6.2 arg_parser.hpp	15
6.3 jacobi_eigensolver.hpp File Reference	15
6.4 jacobi_eigensolver.hpp	16
6.5 problems.hpp File Reference	16
6.5.1 Function Documentation	16
6.5.1.1 problem_5()	16
6.5.1.2 problem_6()	17
6.6 problems.hpp	17
6.7 triDag.hpp File Reference	17
6.8 triDag.hpp	18
6.9 utils.hpp File Reference	18
6.10 utils.hpp	19
Index	21

Chapter 1

Topic Index

1.1 Topics

Here is a list of all topics with brief descriptions:

Standalone Functions	7
--------------------------------	-------------------

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Args	Struct to hold command-line arguments	11
TriDag	Class representing a tridiagonal matrix, i.e. an matrix of the form	13

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

arg_parser.hpp	15
jacobi_eigensolver.hpp	15
problems.hpp	16
triDag.hpp	17
utils.hpp	18

Chapter 4

Topic Documentation

4.1 Standalone Functions

Various stand-alone functions appearing in the project.

Functions

- void [jacobi_rotate](#) (arma::mat &A, arma::mat &R, int k, int l)
Performs a single Jacobi rotation.
- void [jacobi_eigsolver](#) (const arma::mat &A, double eps, arma::vec &eigenvalues, arma::mat &eigenvectors, const int maxiter, int &iterations, bool &converged)
Computes the eigenvalues and eigenvectors of a symmetric matrix using Jacobi's rotation method.
- void [analytic_solution](#) (arma::vec &eigenvalues, arma::mat &eigenvectors, double a, double d, int N)
Gives the analytic solution for the eigenvalues and eigenvectors of $A\vec{v} = \lambda\vec{v}$, where A is a tridiagonal matrix(a,d,a).
- arma::mat [create_tridiagonal](#) (int n, double a, double d, double e)
Creates a symmetric tridiagonal matrix of size $n \times n$ with constant diagonal elements d , sub-diagonal elements a , and super-diagonal elements e .
- double [max_offdiag_symmetric](#) (const arma::mat &A, int &k, int &l)
Finds the greatest off-diagonal element in the upper triangular part (in absolute value) of a given symmetric matrix.

4.1.1 Detailed Description

Various stand-alone functions appearing in the project.

4.1.2 Function Documentation

4.1.2.1 analytic_solution()

```
void analytic_solution (
    arma::vec & eigenvalues,
    arma::mat & eigenvectors,
    double a,
    double d,
    int N)
```

Gives the analytic solution for the eigenvalues and eigenvectors of $A\vec{v} = \lambda\vec{v}$, where A is a tridiagonal matrix(a,d,a).

Parameters

<i>eigenvalues</i>	Armadillo vector for eigenvalues.
<i>eigenvectors</i>	Armadillo vector for eigenvectors.
<i>a</i>	Upper and lower diagonal of matrix.
<i>d</i>	Diagonal of matrix.
<i>N</i>	Size of matrix.

4.1.2.2 `create_tridiagonal()`

```
arma::mat create_tridiagonal (
    int n,
    double a,
    double d,
    double e)
```

Creates a symmetric tridiagonal matrix of size $n \times n$ with constant diagonal elements d , sub-diagonal elements a , and super-diagonal elements e .

Parameters

<i>n</i>	size of the matrix
<i>a</i>	sub-diagonal elements
<i>d</i>	diagonal elements
<i>e</i>	super-diagonal elements

Returns

the tridiagonal matrix

4.1.2.3 `jacobi_eigsolver()`

```
void jacobi_eigsolver (
    const arma::mat & A,
    double eps,
    arma::vec & eigenvalues,
    arma::mat & eigenvectors,
    const int maxiter,
    int & iterations,
    bool & converged)
```

Computes the eigenvalues and eigenvectors of a symmetric matrix using Jacobi's rotation method.

Parameters

<i>A</i>	The symmetric matrix to be diagonalized.
<i>eps</i>	The convergence tolerance for the off-diagonal elements.
<i>eigenvalues</i>	Vector to store the computed eigenvalues (output).
<i>eigenvectors</i>	Matrix to store the computed eigenvectors (output).
<i>maxiter</i>	The maximum number of iterations allowed.
<i>iterations</i>	The number of iterations performed (output).
<i>converged</i>	Boolean flag indicating whether the method converged (output).

4.1.2.4 `jacobi_rotate()`

```
void jacobi_rotate (
    arma::mat & A,
    arma::mat & R,
    int k,
    int l)
```

Performs a single Jacobi rotation.

Parameters

<i>A</i>	The symmetric matrix to be diagonalized.
<i>R</i>	The matrix of eigenvectors.
<i>k</i>	The row index of the maximal off-diagonal element.
<i>l</i>	The column index of the maximal off-diagonal element.

4.1.2.5 `max_offdiag_symmetric()`

```
double max_offdiag_symmetric (
    const arma::mat & A,
    int & k,
    int & l)
```

Finds the greatest off-diagonal element in the upper triangular part (in absolute value) of a given symmetric matrix.

Parameters

<i>A</i>	Symmetric matrix.
<i>k</i>	Row index.
<i>l</i>	Column index.

Returns

Greatest off-diagonal element in the upper triangular part (in absolute value) of A.

Chapter 5

Class Documentation

5.1 Args Struct Reference

Struct to hold command-line arguments.

```
#include <arg_parser.hpp>
```

Public Attributes

- `std::string outfile` = "build/outfile.csv"
Where and what to store outfile, associated to given problem (5 or 6).
- `bool run_tests` = false
If true, runs the tests defined in "tests/".
- `bool run_problem_5` = false
If true, runs problem 5.
- `bool run_problem_6` = false
If true, runs problem 6.
- `double tol` = 1e-14
Tolerance when running Jacobi's rotation method.
- `int n_steps` = 10
Number of steps when running Jacobi's rotation method.
- `int N_max` = 100
Number of different sizes for the matrix A in Jacobi's rotation method (problem 5).
- `int maxiter` = 10000
Maximum number of iterations when running Jacobi's method.

5.1.1 Detailed Description

Struct to hold command-line arguments.

5.1.2 Member Data Documentation

5.1.2.1 maxiter

```
int Args::maxiter = 10000
```

Maximum number of iterations when running Jacobi's method.

5.1.2.2 N_max

```
int Args::N_max = 100
```

Number of different sizes for the matrix A in Jacobi's rotation method (problem 5).

5.1.2.3 n_steps

```
int Args::n_steps = 10
```

Number of steps when running Jacobi's rotation method.

5.1.2.4 outfile

```
std::string Args::outfile = "build/outfile.csv"
```

Where and what to store outfile, associated to given problem (5 or 6).

5.1.2.5 run_problem_5

```
bool Args::run_problem_5 = false
```

If true, runs problem 5.

5.1.2.6 run_problem_6

```
bool Args::run_problem_6 = false
```

If true, runs problem 6.

5.1.2.7 run_tests

```
bool Args::run_tests = false
```

If true, runs the tests defined in "tests/".

5.1.2.8 tol

```
double Args::tol = 1e-14
```

Tolerance when running Jacobi's rotation method.

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

- [arg_parser.hpp](#)

5.2 TriDag Class Reference

Class representing a tridiagonal matrix, i.e. an matrix of the form.

```
#include <triDag.hpp>
```

Public Member Functions

- [TriDag](#) (double h, int N)
Creates a new tridiagonal object.
- void [compute_eigenvalues](#) ()
Computes the eigenvalues of the tridiagonal matrix using arma::eig_sym.
- void [print](#) (int max=25)
Prints the elements of the tridiagonal matrix.

Public Attributes

- arma::mat [A](#)
- arma::vec [eigenvalues](#)
- arma::mat [eigenvectors](#)

5.2.1 Detailed Description

Class representing a tridiagonal matrix, i.e. an matrix of the form.

$$A = \begin{pmatrix} a & c & 0 & \cdots & 0 \\ b & a & c & \cdots & 0 \\ 0 & b & a & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & c \\ 0 & \cdots & 0 & b & a \end{pmatrix},$$

where $b = c = -1/h^2$ and $a = 2/h^2$.

Parameters

<i>A</i>	Armadillo matrix.
<i>eigenvalues</i>	Armadillo vector for the eigenvalues of the matrix A.
<i>eigenvectors</i>	Armadillo vector for the eigenvectors of the matrix A.

See also

[TriDag](#)

5.2.2 Constructor & Destructor Documentation

5.2.2.1 TriDag()

```
TriDag::TriDag (
    double h,
    int N)
```

Creates a new tridiagonal object.

Parameters

<i>h</i>	Stepsize, defining the diagonals of the tridiagonal matrix, see general description .
<i>N</i>	Size of matrix.

5.2.3 Member Function Documentation

5.2.3.1 compute_eigenvalues()

```
void TriDag::compute_eigenvalues ()
```

Computes the eigenvalues of the tridiagonal matrix using arma::eig_sym.

5.2.3.2 print()

```
void TriDag::print (
    int max = 25)
```

Prints the elements of the tridiagonal matrix.

Parameters

<i>max</i>	Only prints matrix if the size of the matrix is less than or equal to max.
------------	--

5.2.4 Member Data Documentation

5.2.4.1 A

```
arma::mat TriDag::A
```

5.2.4.2 eigenvalues

```
arma::vec TriDag::eigenvalues
```

5.2.4.3 eigenvectors

```
arma::mat TriDag::eigenvectors
```

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

- [triDag.hpp](#)

Chapter 6

File Documentation

6.1 arg_parser.hpp File Reference

```
#include <string>
#include <iostream>
```

Include dependency graph for arg_parser.hpp:

6.2 arg_parser.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef ARG_PARSER_CPP
00002 #define ARG_PARSER_CPP
00003
00004 #include <string>
00005 #include <iostream>
00006
00010 struct Args
00011 {
00012     std::string outfile = "build/outfile.csv";
00013     bool run_tests = false;
00014     bool run_problem_5 = false;
00015     bool run_problem_6 = false;
00016     double tol = 1e-14;
00017     int n_steps = 10;
00018     int N_max = 100;
00019     int maxiter = 10000;
00020 };
00021
00022
00029 Args parse_args(int argc, char *argv[]);
00030
00031 #endif
```

6.3 jacobi_eigensolver.hpp File Reference

```
#include <armadillo>
#include "utils.hpp"
```

Include dependency graph for jacobi_eigensolver.hpp:

Functions

- void [jacobi_rotate](#) (arma::mat &A, arma::mat &R, int k, int l)
Performs a single Jacobi rotation.
- void [jacobi_eigsolver](#) (const arma::mat &A, double eps, arma::vec &eigenvalues, arma::mat &eigenvectors, const int maxiter, int &iterations, bool &converged)
Computes the eigenvalues and eigenvectors of a symmetric matrix using Jacobi's rotation method.

6.4 jacobi_eigsolver.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef JACOBI_EIGENSOLVER_HPP
00002 #define JACOBI_EIGENSOLVER_HPP
00003
00004 #include <armadillo>
00005 #include "utils.hpp"
00006
00010
00019 void jacobi_rotate(arma::mat &A, arma::mat &R, int k, int l);
00020
00032 void jacobi_eigsolver(const arma::mat &A, double eps, arma::vec &eigenvalues, arma::mat
&eigenvectors, const int maxiter, int &iterations, bool &converged);
00033
00034 #endif
00035
```

6.5 problems.hpp File Reference

```
#include <string>
```

Include dependency graph for problems.hpp:

Functions

- void [problem_5](#) (double N_max, double tol, int maxiter, const std::string &outfile)
Iterates from $N = 5$ to N_{max} , and for each iteration creating a symmetric $N \times N$ matrix using `triDag::create_↵_tridiagonal` and computing the eigenvalues using Jacobi's rotation method implemented in `jacobi_eigsolver::jacobi_↵_eigsolver`. Writes the result to `outfile`.
- void [problem_6](#) (int n_steps, double tol, int maxiter, const std::string &outfile)
Creates a tridiagonal matrix using `triDag::create_tridiagonal`, computes its eigenvalues and eigenvectors using Jacobi's rotation method implemented in `jacobi_eigsolver::jacobi_eigsolver`. Writes these eigenvalues and eigenvectors to `outfile`.

6.5.1 Function Documentation

6.5.1.1 problem_5()

```
void problem_5 (
    double N_max,
    double tol,
    int maxiter,
    const std::string & outfile)
```

Iterates from $N = 5$ to N_{max} , and for each iteration creating a symmetric $N \times N$ matrix using `triDag::create_↵_tridiagonal` and computing the eigenvalues using Jacobi's rotation method implemented in `jacobi_eigsolver_↵::jacobi_eigsolver`. Writes the result to `outfile`.

Parameters

<i>N_max</i>	Final size of matrix.
<i>tol</i>	Tolerance passed to <code>jacobi_eigensolver::jacobi_eigensolver</code> .
<i>maxiter</i>	Maximum number of iterations.
<i>outfile</i>	File to write results to.

6.5.1.2 problem_6()

```
void problem_6 (
    int n_steps,
    double tol,
    int maxiter,
    const std::string & outfile)
```

Creates a tridiagonal matrix using `triDag::create_tridiagonal`, computes its eigenvalues and eigenvectors using Jacobi's rotation method implemented in `jacobi_eigensolver::jacobi_eigensolver`. Writes these eigenvalues and eigenvectors to `outfile`.

Parameters

<i>n_steps</i>	Number if steps for Jacobi's rotation method (1 - size of tridiagonal matrix).
<i>tol</i>	Tolerance passed to <code>jacobi_eigensolver::jacobi_eigensolver</code> .
<i>maxiter</i>	Maximum number of iterations.
<i>outfile</i>	File to write results to.

6.6 problems.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef PROBLEMS
00002 #define PROBLEMS
00003 #include <string>
00004
00015 void problem_5(double N_max, double tol, int maxiter, const std::string &outfile);
00016
00017
00027 void problem_6(int n_steps, double tol, int maxiter, const std::string &outfile);
00028
00029 #endif
```

6.7 triDag.hpp File Reference

```
#include <armadillo>
```

Include dependency graph for `triDag.hpp`:

Classes

- class `TriDag`

Class representing a tridiagonal matrix, i.e. an matrix of the form.

Functions

- void [analytic_solution](#) (arma::vec &eigenvalues, arma::mat &eigenvectors, double a, double d, int N)
Gives the analytic solution for the eigenvalues and eigenvectors of $A\vec{v} = \lambda\vec{v}$, where A is a tridiagonal matrix(a,d,a).

6.8 triDag.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef TRI_DAG
00002 #define TRI_DAG
00003 #include <armadillo>
00004
00005
00023 class TriDag{
00024 private:
00025     double a_fill;
00026     double d_fill;
00027     int N;
00028
00029     void create_matrix(); // Creates tridiagonal matrix
00030
00031 public:
00032     arma::mat A;
00033     arma::vec eigenvalues;
00034     arma::mat eigenvectors;
00035
00042     TriDag(double h, int N);
00043
00044
00049     void compute_eigenvalues();
00050
00056     void print(int max=25);
00057 };
00058
00072 void analytic_solution(arma::vec &eigenvalues, arma::mat &eigenvectors, double a, double d, int N);
00074
00075 #endif
```

6.9 utils.hpp File Reference

```
#include <armadillo>
```

Include dependency graph for utils.hpp: This graph shows which files directly or indirectly include this file:

Functions

- arma::mat [create_tridiagonal](#) (int n, double a, double d, double e)
Creates a symmetric tridiagonal matrix of size $n \times n$ with constant diagonal elements d , sub-diagonal elements a , and super-diagonal elements e .
- double [max_offdiag_symmetric](#) (const arma::mat &A, int &k, int &l)
Finds the greatest off-diagonal element in the upper triangular part (in absolute value) of a given symmetric matrix.

6.10 utils.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef UTILS
00002 #define UTILS
00003 #include <armadillo>
00004
00005
00010
00011
00012
00016
00026 arma::mat create_tridiagonal(int n, double a, double d, double e);
00027
00028
00037 double max_offdiag_symmetric(const arma::mat &A, int &k, int &l);
00038
00040 #endif
```


Index

A

- TriDag, [14](#)
- analytic_solution
 - Standalone Functions, [7](#)
- arg_parser.hpp, [15](#)
- Args, [11](#)
 - maxiter, [12](#)
 - N_max, [12](#)
 - n_steps, [12](#)
 - outfile, [12](#)
 - run_problem_5, [12](#)
 - run_problem_6, [12](#)
 - run_tests, [12](#)
 - tol, [12](#)
- compute_eigenvalues
 - TriDag, [14](#)
- create_tridiagonal
 - Standalone Functions, [8](#)
- eigenvalues
 - TriDag, [14](#)
- eigenvectors
 - TriDag, [14](#)
- jacobi_eigensolver
 - Standalone Functions, [8](#)
- jacobi_eigensolver.hpp, [15](#), [16](#)
- jacobi_rotate
 - Standalone Functions, [8](#)
- max_offdiag_symmetric
 - Standalone Functions, [9](#)
- maxiter
 - Args, [12](#)
- N_max
 - Args, [12](#)
- n_steps
 - Args, [12](#)
- outfile
 - Args, [12](#)
- print
 - TriDag, [14](#)
- problem_5
 - problems.hpp, [16](#)
- problem_6
 - problems.hpp, [17](#)
- problems.hpp, [16](#), [17](#)
 - problem_5, [16](#)
 - problem_6, [17](#)
- run_problem_5
 - Args, [12](#)
- run_problem_6
 - Args, [12](#)
- run_tests
 - Args, [12](#)
- Standalone Functions, [7](#)
 - analytic_solution, [7](#)
 - create_tridiagonal, [8](#)
 - jacobi_eigensolver, [8](#)
 - jacobi_rotate, [8](#)
 - max_offdiag_symmetric, [9](#)
- tol
 - Args, [12](#)
- TriDag, [13](#)
 - A, [14](#)
 - compute_eigenvalues, [14](#)
 - eigenvalues, [14](#)
 - eigenvectors, [14](#)
 - print, [14](#)
 - TriDag, [14](#)
- triDag.hpp, [17](#), [18](#)
- utils.hpp, [18](#), [19](#)