Project 2

For the course FYS3150

Erik Grammeltvedt, Erlend Tiberg North and Alexandra Jahr Kolstad

September 29, 2019

Week 37 - 40

Contents

1 Abstract

bla bla bla bla bla bla bla

2 Introduction

All programs are found at our GitHub-repository.

3 Method

Hvilke filer som er til hvilke oppgaver

Our project consists of the files `jacobimethod.cpp` and `plot_data.py`. For exercises **b)** through **e)** we use the file `jacobimethod.cpp`. Also for exercise **b)** we have the file `plot_data.py`.

3.1 Exercise a)

In this exercise we are going to prove that $\vec{w}_i = U\vec{v}_i$ is an orthogonal or unitary transformation that preserves the dot product and orthogonality. We start by multiplying $\vec{w}_j^T$ with $\vec{w}_i$ to take the vector product, also called the dot product. If the vector product of these vectors is equal to δ_{ij}, given by $\vec{v}_j^T\vec{v}_i = \delta_{ij}$ in the exercise, then the dot product and orthogonality is preserved. In this exercise we assume that $U^TU = I$, where I is the identity matrix, because this defines a unitary matrix U which we compute with in this exercise.

The vector product is calculated as followed:

$$\begin{aligned}
\vec{w}_j^T\vec{w}_i &= (U\vec{v})^T U\vec{v}_i \\
&= \vec{v}_j^T U^T U\vec{v}_i \\
&= \vec{v}_j^T\vec{v}_i \\
&= \delta_{ij}
\end{aligned}$$

The vector product of $\vec{w}_j^T$ and $\vec{w}_i$ is δ_{ij}, which proves that the dot product and orthogonality is preserved for the transformation.

3.2 Exercise b)

3.2.1 Calculations

In this prject we compute with a symmetric matrix, similar to the matrix **A** in project 1. This matrix is given by the matrix equation

$$\begin{bmatrix} d & a & 0 & \dots & 0 & 0 \\ a & d & a & \dots & 0 & 0 \\ 0 & a & d & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & a & d & a \\ 0 & 0 & 0 & 0 & a & d \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix}$$

where $d = \frac{2}{h^2}$ and $a = -\frac{1}{h^2}$. We implement these values later in exercise **d)**. For now in this exercise, we have set $d = 2$ and $a = -1$. **still correct??**

λ are eigenvalues given by the equation

$$\lambda_j = d + 2a\cos\left(\frac{j\pi}{N+1}\right)$$

given for $j = 1, 2, ..., N$.
Skal kommentere noe for egenverdi og egenvektor?

3.2.2 The programming

Commenting the code `jacobimethod.cpp`. OBS.

The program starts with defining a function for finding the max values of the off-diagonal elements. This is the function `offdiag`, which is taken from the lecture notes. The same applies for the function `Jacobi_rotate`. `Jacobi_rotate` is the function for rotating and computing the matrix. It calculates the equation

$$\mathbf{B} = \mathbf{S}^T A \mathbf{S}$$

for finding the diagonal matrix with the eigenvalues of the matrix **A**. It also computes the eigenvectors, and stores them in a matrix **R**.

In the main function we define the matrix **A** given by the constants d and a. See 3.2.1 for the definitions of the matrix. Then the clock is set to start, and it times how long the armadillo

The program `plot_data.py` reads the `.txt`-file made in `jacobimethod.cpp` and plots the data. `jacobimethod.cpp` generates the file `stats.txt`, which contains the dimension of the matrices, `n`, the number of iterations, `i`, the time used for the Armadillo-funtion `eig_syl`, `timespan eig_sym` and the time used in our algorithm, `timespan ours`. `plot_data.py` plots the number of iterations needed given by different values of the matrix dimension and it plots the time needed as a function of the matrix dimensions. The values in `stats.txt` is taken from multiple runs of `jacobimethod.cpp` for different dimensions of the matrices. The figures (1) and (2) are the plots from `plot_data.py`. **HAR FLERE PLOT Å PLOTTE**

Skal egentlig kommentere dette?????

The number of similarity transformations needed to reach the desired matrix depends on the dimension n. For instance a run of our matrix $\mathbf{A}$ given as a (10×10) matrix, there are 154 transformations needed. This number is only exact for this specific run, as it will change for any differences to the matrix, both size and elements.

In the lecture notes it states that for the Jacobi method there is no way to predict the number of transformations needed. See this file under *Discussion for Householder's method for eigenvalues.* **Riktig??**

3.3 Exercise c)

3.3.1 Calculations

3.3.2 The programming

3.4 Exercise d)

3.4.1 Calculations

3.4.2 The programming

3.5 Exercise e)

3.5.1 Calculations

3.5.2 The programming

4 Results and discussion

Our results are as shown in the Appendix. We also have `.txt`-files for all the raw data generated by the projects up on GitHub.

4.1 Exercise a)

The results are given in the method for **a)**, see 3.1.

4.2 Exercise b)

Under follows the data in `stats.txt`.

```
n, iterations, timespan eig_sym, timespan ours
3, 10, 1.137520e-04, 1.666000e-05
5, 32, 8.632200e-05, 6.015200e-05
10, 154, 8.277800e-05, 3.797710e-04
15, 363, 1.395090e-04, 1.351192e-03
20, 644, 2.044150e-04, 3.183269e-03
25, 1025, 2.189260e-04, 7.074493e-03
30, 1463, 1.089399e-02, 1.494651e-02
40, 2685, 1.947745e-02, 4.040845e-02
50, 4115, 8.989085e-03, 8.897613e-02
60, 6007, 7.917790e-04, 1.650397e-01
70, 8081, 1.022618e-02, 2.948578e-01
80, 10487, 1.577296e-03, 4.732270e-01
90, 13338, 2.079035e-03, 7.397605e-01
100, 16438, 1.184104e-02, 1.103670e+00
110, 19905, 2.845618e-03, 1.580353e+00
120, 23547, 3.005831e-03, 2.222943e+00
130, 27615, 3.534705e-03, 2.997598e+00
140, 31981, 4.432247e-03, 4.003528e+00
150, 36537, 4.577840e-03, 5.204313e+00
160, 41531, 5.158932e-03, 6.712134e+00
170, 47005, 6.375916e-03, 8.584403e+00
180, 52424, 7.180886e-03, 1.065379e+01
190, 58289, 7.780051e-03, 1.319042e+01
200, 64379, 8.258574e-03, 1.607514e+01
```

We observe that for small matrix-dimensions n our algorithm is sligthly faster than the Armadillo function `eig_sym`. When n increases in value, the time used in our algorithm increases exponentially. For the biggest given dimension, $n = 200$, our algorithm uses $16s$ while `eig_sym` uses only $0.008s$. Here we can observe how slow
Koden bruker lengre tid både fordi matrisen er større, slik at det er mer å rotere og fordi den må utføre flere iterasjoner.

Hvorfor er jacobimethod tregere enn eigsym? økende n, jacobi øker verdien til noen elementer, mens den minker noen andre verdier. altså den fucker noen verdier, mens den fikser noen andre.

eigsym bruker ikke jacobi metoden, og kan dermed være raskere. den kan f.eks. bruke thomas algorithm, som vi programerte i project 1, som er raskere for større matriser. tror den bruker en if-else-statement med ulike algoritmer til å løse de forskjellige matrisene - har prøvd å finne kildefilen gitt ved armadillo, men den finner jeg ikke.

når man skriver inn std istedenfor dc for eigsym er dc litt raskere for store matriser, med n rundt 200, men siden vi

4.3 Exercise c)

4.4 Exercise d)

4.5 Exercise e)

5 Conclusion and perspective

6 Appendix

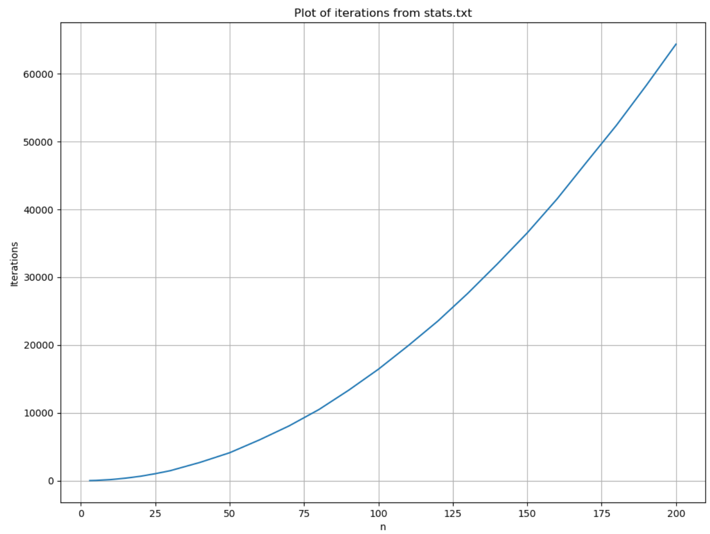

Figure 1: The plot of iterations for the Jacobi method as function of the dimension n of the matrix $\mathbf{A}$.

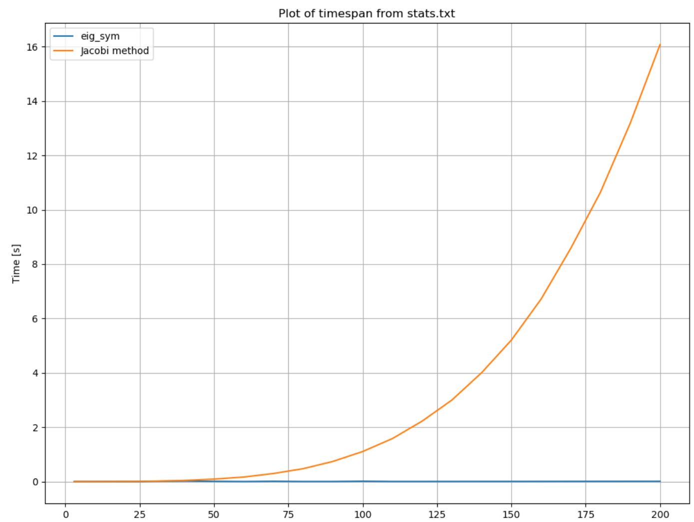

Figure 2: The plot of the time the function `eig_sym` from Armadillo uses and the time Jacobi method uses as functions of the dimension n if the matrix $\mathbf{A}$.

7 References

Link to the PDF for Project 2.

Our GitHub-repository.

Link to lecture slides in FYS3150 - Computational Physics.