A Jacobi Algorithm for Simultaneous diagonalization of Several Symmetric Matrices

by

Mercy Maleko

Master's Thesis in Scientific Computing
Department of Numerical Analysis and Computer Science
Royal Institute of Technology
Stockholm 2003
Presented on:20th January 2003.

Abstract

A Jacobi algorithm that will diagonalize simultaneous more than one dense symmetric matrices is developed in this thesis. The development is based on the original Jacobi algorithm that diagonalize one matrix at a time, by performing Jacobi rotations until the matrix is diagonal to machine precission. The algorithm will be modified so as to solve the Singular Value Decomposition for several dense symmetric matrices simultaneously.
First, we will find the algorithm that will diagonalize the matrices, then test the algorithm with different types of matrices.
Second, modify the algorithm to solve the Singular value problems. We will test it with matrices obtained from some experiments performed to determine chemical composition of a sample mixture. This sample mixture contains some species, the number of non zero singular values will give the number of species contained in the mixture.

Sammanfattning

I detta examensarbete konstreras en Jacobi algoritm som diagonaliserar flera glesa symmetriska matriser samtidigt. Den bygger på Jacobi algoritm för diagonalisering av en matris genom Jacobirotationer tills matrisen är diagonal upp till maskinnoggrannhet. Algoritmen modifieras för att lösa singulärvärdes uppdelningen av flera matriser samtidigt.
Först, konstruerar vi algoritmen för samtidig diagonalisering av flera matriser fäljt av tester på olika typer av matriser.
Sedan, modifierar vi algoritmen för att lösa singulärvärdes problem. Tester genomförs på matriser fråm experiment utförda för att bestämma kemisk sammansättning av en lösning. Lösningen innehåller ämnen och antalet ämnen ges av olika antalet nollskilsda singulärvärden.

Acknowledgements

I first wish to thank my supervisor, Prof. Axel Ruhe, for his continuous support and continuous and enlightening guidance during completion and writting of my thesis.
I am also grateful to Dr. Lennart Edsberg (Associate Prof.), for being an excellent coordinator for International Masters program at Nada, and for the fruitfull contacts we had made when I was applying this program.
I owe many thanks to my sponsor, The Swedish Institute (SI), and in particular to Ms Karin Diff.
I would like to thank my office mates, Per-Olov, Reynir Gudmundsson and Henrik Olsson for their advises and help. Also thanks to my colleague for their day to day activities we shared during course work.
My heartly thanks have to go to my family and relatives for their support.
Lastly but not least, I want to specially thank Dr.H.M Twaakyondo, for his endless encouragement during the pursuit of my Masters degree and the composition of the thesis.

Contents

To Dr. Hashim M. Twaakyondo

Chapter 1

Introduction

The method of Jacobi dates back to 1846 before the advent of the QR algorithm. The method solves the eigenvalue problem for real, dense symmetric matrices. The Jacobi algorithm starts by reducing the dense matrix $\mathbf{A}$ into diagonal matrix $\mathbf{A}'$ whose diagonal elements are the eigenvalues of the original matrix $\mathbf{A}$. It employs a sequence of orthogonal similarity transformations. Each transformation is a plane rotation designed to annihilate one of the off diagonal matrix elements. In each successive transformation, the off diagonal elements are changed and each time they get smaller and smaller until the matrix is diagonal to machine precision. It is one of the backward stable method and thus compute large eigenvalues (those near $\|A\|_2$ in magnitude) with high relative accuracy.

Jacobi algorithm has been used to diagonalize symmetric matrix by performing a basic Jacobi plane rotation $R(\theta)$, given as

$$\mathbf{R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{1.1}$$

The matrix $\mathbf{A}$ can be 2x2 or more, Jacobi method solves a sequence of 2x2 subproblems.
Given the 2x2 subproblem (i, j),

$$\mathbf{A} = \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{pmatrix}$$

A transformed matrix $\mathbf{A}'$ will be

$$\mathbf{A}' = \ R(i, j, \theta)^T A \ R(i, j, \theta) \tag{1.2}$$

with

$$\mathbf{R}(\mathbf{i},\mathbf{j},\theta) = \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & \cos\theta & & & & -\sin\theta & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & 1 & & & \\ & & & \sin\theta & & & & \cos\theta & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & & & & & & & & & & 1 \end{pmatrix} \tag{1.3}$$

The angle θ is chosen so that the off diagonal elements $a_{ij} = a_{ji}$ are reduced to zero. Note that only the entries in columns and rows i and j of $\mathbf{A}$ will change. Since the matrix is symmetric, only the upper triangular part of $\mathbf{A}$ needs to be computed. Jacobi's method is usually slower than Householder, QR that is used in Matlab, but it remains of interest because:-

- It can sometimes compute tiny eigenvalues and their eigenvectors with high relative accuracy.
- It is an efficient method for computing eigenvalues of a *nearly diagonal matrix*.
- It can be used to diagonalize several symmetric matrices simultaneously, or compute an approximate diagonalization when the matrices do not commute exactly. This is the main focus of this project.

In the second chapter of this thesis, we are going to explain about the original Jacobi, the Jacobi Algorithm for one symmetric matrix, we call it algorithm J1. In the same chapter we will discuss some conditions for convergence for this algorithm. Chapter 3 will deal with the whole idea of diagonalization of several matrices simultaneously. Here we will derive the algorithm for diagonalization of several symmetric matrices simultaneously, and also we will have a look on what conditions are suitable for convergence. Chapter 4 discusses about the application of the Jacobi algorithm to SVD, how we can apply the algorithm found to singular value decomposition problems. Moreover, in chapter 5, we will show some numerical tests we have performed in testing the algorithm.

Chapter 2

Jacobi Algorithm for one Symmetric Matrix

As highlighted in chapter 1, that Jacobi method consists of a sequence of orthogonal similarity transformations. In this chapter we are going to discuss the Jacobi Algorithm as applied to a single matrix, this algorithm can also be found in different advance Linear Algebra books [1] [5] [7].

2.1 Formulation

To construct Jacobi transformation $R(i,j,\theta)$ on a matrix $\mathbf{A}$ we take

$$\mathbf{A}' = \ R(i,j,\theta)^T A \ R(i,j,\theta) \tag{2.1}$$

$$= \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \quad \begin{bmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{bmatrix} \quad \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta\ a_{ii} + \sin\theta\ a_{ji} & \cos\theta\ a_{ij} + \sin\theta\ a_{jj} \\ -\sin\theta\ a_{ii} + \cos\theta\ a_{ji} & -\sin\theta\ a_{ij} + \cos\theta\ a_{jj} \end{bmatrix} \quad \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} c^2\ a_{ii} + s^2\ a_{jj} + sc\ a_{ij} + sc\ a_{ji} & -cs\ a_{ii} + cs\ a_{jj} + c^2\ a_{ij} - s^2\ a_{ji} \\ -sc\ a_{ii} + c^2\ a_{ji} - s^2\ a_{ij} + cs\ a_{jj} & s^2\ a_{ii} - cs\ a_{ji} - cs\ a_{ij} + c^2\ a_{jj} \end{bmatrix}$$

Since $\mathbf{A}$ is real and symmetric, $a_{ij} = a_{ji}$.

then

$$\mathbf{A}' = \begin{bmatrix} c^2\ a_{ii} + s^2\ a_{jj} + 2sc\ a_{ij} & cs\ (a_{jj} - a_{ii}) + (c^2 - s^2)a_{ij} \\ cs\ (a_{jj} - a_{ii}) + (c^2 - s^2)\ a_{ij} & s^2\ a_{ii} - 2cs\ a_{ij} + c^2\ a_{jj} \end{bmatrix} \tag{2.2}$$

where $c = \cos(\theta)$ and $s = \sin(\theta)$

Expressing $\mathbf{A}'$ as

$$\mathbf{A}' = \begin{pmatrix} a'_{ii} & a'_{ij} \\ a'_{ji} & a'_{jj} \end{pmatrix}$$

we have

$$a'_{ij} = (\cos^2\theta - \sin^2\theta)\ a_{ij} + \cos\theta\sin\theta\ (a_{jj} - a_{ii}) \tag{2.3}$$

$$a'_{ij} = \cos 2\theta\ a_{ij} + 0.5\sin 2\theta\ (a_{jj} - a_{ii}) \tag{2.4}$$

The 2×2 matrix will be diagonal if

$a'_{ij} = 0$,

that is if

$$\tan\ 2\theta = \frac{-2\ a_{ij}}{(a_{jj} - a_{ii})}. \tag{2.5}$$

The transformed diagonal elements a'_{ii} and a'_{jj} are as follows

$$a'_{ii} = \cos^2\theta a_{ii} + \sin^2\theta a_{jj} + 2\sin\theta\ \cos\theta a_{ij} \tag{2.6}$$

$$a'_{jj} = \sin^2\theta\ a_{ii} - 2\cos\theta\ \sin\theta a_{ij} + \cos^2\theta a_{jj} \tag{2.7}$$

From equation (2.5), we compute $\sin\theta$ and $\cos\theta$ from $\tan 2\theta$ by using the quadratic equation below

$$t^2 + 2\gamma\ t - 1 = 0 \tag{2.8}$$

$$\gamma = \frac{1}{\tan 2\theta}$$

and so

$$t = \frac{\text{sign}(\gamma)}{(|\gamma| + \sqrt{1+\gamma^2})} \tag{2.9}$$

then

$$c = 1/\sqrt{1+t^2},$$
$$s = ct.$$

$t = \tan\theta, s = \sin\theta$ and $c = \cos\theta$

this choice of θ can be shown to minimize the difference

$$\|A' - A\|_F$$

and in particular, this will prevent the exchange of the two diagonal elements a_{ii} and a_{jj} when a_{ij} is small, which is critical the convergence of Jacobi method.

Hence the Jacobi Algorithm for a single matrix is given as follows

ALGORITHM J1

Given A matrix, n size of the matrix, Set V=I
for i=1:n-1
for j=i+1:n
Find angle θ that transforms $a(i,j)$ to zero.
Get the rotation matrix $R(i,j,\theta)$
Rotate the matrix

$$A' = R(i,j,\theta)^T A R(i,j,\theta)$$

accumulate transformations

$$V' = VR(i,j,\theta)$$

endfor
endfor

2.2 Convergence and Stopping criteria

The convergence of Jacobi method can be analysed by considering the sum of squares of the off diagonal elements.

$$\mathbf{S} = \sum_{i \neq j} |a_{ij}|^2$$

Equations (2.3), (2.6), and (2.7) implies that

$$\mathbf{S}' = \mathbf{S} - 2\ |a_{ij}|^2$$

i.e sum of squares of off diagonal elements decreases correspondingly by $2\ |a_{ij}|^2$.

The matrix $\mathbf{A}'$ as said, is diagonal to machine precision, when $\mathbf{S} \leq (eps)^2 \|\mathbf{A}\|^2{}_F$, the diagonal elements are the eigenvalues of $\mathbf{A}$, since

$$\mathbf{A}' = \mathbf{V^T A V} \tag{2.10}$$

where

$$\mathbf{V} = R(i,j,\theta)_1 R(i,j,\theta)_2 R(i,j,\theta)_3 \ldots$$

$R(i,j,\theta)_k$ being the successive Jacobi rotation matrices and the columns of $\mathbf{V}$ are the eigenvectors, since

$$\mathbf{V A}' = \mathbf{A V} \tag{2.11}$$

2.2.1 Stopping criteria

The stopping criteria for Jacobi Algorithm is a small off diagonal quantity such that

$$|a_{ij}| < \delta(\max(|a_{ii}|, |a_{jj}|))$$

2.3 How to choose an element

At each step in the Jacobi transformation, an element is annihilated. To choose which element to be annihilated, there are several strategies.

2.3.1 Classical Jacobi

Based on searching the whole upper triangle at each stage and set the largest off diagonal element to zero. This is reasonable strategy for hand calculation, but it is prohibitive on a computer since the search alone makes each Jacobi rotation a process of order N^2 instead of N. N is given by $\frac{n(n-1)}{2}$, n is the matrix dimension.

2.3.2 Cyclic Jacobi

Based on innihilating elements in a predetermined order regardless of the size. Each element is being rotated exactly once in any sequence of N rotations called a **sweep**. The convergence is generally quadratic. In cyclic Jacobi, ordering can be row-wise scheme, column-wise scheme or diagonal-wise scheme. The most popular cyclic ordering is the row-wise scheme.

In the **row wise ordering scheme** the rotations are performed in the following order.

$$\begin{array}{ccccccc} (1,2) & (1,3) & (1,4) & \dots & (1,n) \\ & (2,3) & (2,4) & \dots & (2,n) \\ & & & \dots & \dots \\ & & & & (n-1,n) \end{array} \tag{2.12}$$

which is repeated cyclically.

Chapter 3

Jacobi Algorithm on several Symmetric Matrices

As we have seen in chapter 2, Jacobi algorithm is used to find the eigenvalues and eigenvectors of a dense symmetric matrix, by making all the off diagonal elements equal to zero and left with diagonal matrix whose diagonal elements are the eigenvalues of the matrix. Also we saw how orthogonal eigenvectors can be obtained by accumulating the transformations. Algorithm J1.

When dealing with more than one dense matrix, it is difficult to annihilate all the off diagonal elements, instead the best way is to minimize the sum of squares of their off diagonal elements. To diagonalize them simultaneously, first we will have to find the angle θ from the diagonal elements and off diagonal elements of all the matrices involved. This angle θ will be used by all matrices to transform them to diagonal form.

3.1 Formulation

Given for example, three real 2×2 symmetric matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, then according to equation (2.1).

$$\begin{aligned}
\mathbf{A}' &= R(i,j,\theta)^T A\ R(i,j,\theta) \\
\mathbf{B}' &= R(i,j,\theta)^T B\ R(i,j,\theta) \\
\mathbf{C}' &= R(i,j,\theta)^T C\ R(i,j,\theta)
\end{aligned}$$

by equation (2.2)

$$\mathbf{A}' = \begin{bmatrix} c^2\ a_{ii} + s^2\ a_{jj} + 2sc\ a_{ij} & cs\ (a_{jj} - a_{ii}) + (c^2 - s^2)a_{ij} \\ cs\ (a_{jj} - a_{ii}) + (c^2 - s^2)\ a_{ij} & s^2\ a_{ii} - 2cs\ aij + c^2\ a_{jj} \end{bmatrix} \tag{3.1}$$

$$\mathbf{B}' = \begin{bmatrix} c^2\ b_{ii} + s^2\ b_{jj} + 2sc\ b_{ij} & cs\ (b_{jj} - b_{ii}) + (c^2 - s^2)b_{ij} \\ cs\ (b_{jj} - b_{ii}) + (c^2 - s^2)\ b_{ij} & s^2\ b_{ii} - 2cs\ bij + c^2\ b_{jj} \end{bmatrix} \tag{3.2}$$

$$\mathbf{C}' = \begin{bmatrix} c^2\ c_{ii} + s^2\ c_{jj} + 2cs\ c_{ij} & cs\ (c_{jj} - c_{ii}) + (c^2 - s^2)c_{ij} \\ cs\ (c_{jj} - c_{ii}) + (c^2 - s^2)\ c_{ij} & s^2\ c_{ii} - 2cs\ cij + c^2\ c_{jj} \end{bmatrix} \tag{3.3}$$

the off diagonal elements will be expressed as

$$a'_{ij} = (c^2 - s^2)\ a_{ij} + cs\ (a_{jj} - a_{ii}) \tag{3.4}$$

$$b'_{ij} = (c^2 - s^2)\ b_{ij} + cs\ (b_{jj} - b_{ii}) \tag{3.5}$$

$$c'_{ij} = (c^2 - s^2)\ c_{ij} + cs\ (c_{jj} - c_{ii}) \tag{3.6}$$

Generally, the sum of squares of off diagonal elements is given by

$$SS = \sum_{k=1}^{p} \sum_{i \neq j}^{n} ({a'_{ij}}^2)_k \qquad k = 1, 2, 3 \ldots p \ \text{ the number of matrices involved.} \tag{3.7}$$

So sum of squares for $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ will be

$$SS = {a'_{ij}}^2 + {b'_{ij}}^2 + {c'_{ij}}^2 \tag{3.8}$$

substituting equations 3.4-3.6 on 3.8, we have

$$SS = \cos^2 2\theta D + (1/4)\sin^2 2\theta E + \sin 2\theta \cos 2\theta F \tag{3.9}$$

where

$$D = a_{ij}^2 + b_{ij}^2 + c_{ij}^2 \tag{3.10}$$

$$E = (a_{jj} - a_{ii})^2 + (b_{jj} - b_{ii})^2 + (c_{jj} - c_{ii})^2 \tag{3.11}$$

$$F = a_{ij}(a_{jj} - a_{ii}) + b_{ij}(b_{jj} - b_{ii}) + c_{ij}(c_{jj} - c_{ii}) \tag{3.12}$$

Hence equation 3.9 gives the sum of squares of off diagonal elements of the three matrices.
Assuming we now have matrices A, B, C ... P, we define

$$v_{ij} = \begin{pmatrix} a_{ij} \\ b_{ij} \\ c_{ij} \\ \vdots \\ p_{ij} \end{pmatrix} \qquad v_{ii} = \begin{pmatrix} a_{ii} \\ b_{ii} \\ c_{ii} \\ \vdots \\ p_{ii} \end{pmatrix} \qquad v_{jj} = \begin{pmatrix} a_{jj} \\ b_{jj} \\ c_{jj} \\ \vdots \\ p_{jj} \end{pmatrix} \tag{3.13}$$

where v_{ij} is a vector containing the off diagonal elements of each matrix, v_{ii} is a vector with upper diagonal elements of each matrix and v_{jj} is a vector with lower diagonal elements. Then from equations 3.10-3.12 and 3.13 we have

$$\begin{aligned} D &= v_{ij}^T v_{ij} = \|v_{ij}\|^2 \\ E &= \|v_{jj} - v_{ii}\|^2 \\ F &= v_{ij}^T (v_{jj} - v_{ii}) \end{aligned}$$

Using the fact that

$$\cos 4\theta = \cos^2 2\theta - \sin^2 2\theta.$$

and

$$\sin 4\theta = 2\cos 2\theta \sin 2\theta$$

The sum of squares SS from equation 3.9 will now be expressed as

$$SS = \cos 4\theta \left[\frac{D - E/4}{2}\right] + \left[\frac{D + E/4}{2}\right] + \left(\frac{F}{2}\right)\sin 4\theta$$

To find the angle θ that will transform each matrix closest to diagonal form simultaneously we let

$$s = \left[\frac{D - E/4}{2}\right] \qquad g = \left(\frac{F}{2}\right) \qquad \text{and} \qquad k = \left[\frac{D + E/4}{2}\right]$$

This implies that

$$SS = s \times \cos 4\theta + g \times \sin 4\theta + k$$

in amplitude-phase angle form, SS is expressed as

$$SS = r\sin(4\theta + \phi) + k$$
$$\text{with} \quad r = \sqrt{s^2 + g^2}$$

ϕ is the phase angle and is given as

$$\phi = \tan^{-1}(\frac{g}{s}) \quad \text{if } s > 0$$
$$\phi = \pi + \tan^{-1}(\frac{g}{s}) \quad \text{if } s < 0$$

With sine function, SS is minimum when either $\theta = \frac{1}{4}(\frac{3\pi}{2} - \phi)$ or

$\theta = \frac{1}{4}(\frac{-\pi}{2} - \phi)$.

The Jacobi rotation matrix $R(\theta)$ is obtained from θ, and the matrices are transformed to diagonal form, see algorithm J1. The convergence is quadratic based on Frobenius norm

$$\|A' - A\|_F = \big[\sum_{i \neq j} SS_{ij}\big]^{1/2}$$

$$SS = {a'_{ij}}^2 + {b'_{ij}}^2 + {c'_{ij}}^2 \ldots + {p'_{ij}}^2$$

Therefore, in a nutshell the following is the algorithm for diagonalization of several matrices simultaneously.

ALGORITHM J2.

Given $A_1, A_2 \ldots A_p$
Get n, size of the matrix A_k all matrices must be of the same size.
Get the coordinates i, j *of each matrix* $A_1, A_2 \ldots A_p$, *Set V=I*
for i=1:n-1
for j=i+1:n
Find the angle θ *that transforms all off diagonal elements of all p matrices.*
Get the rotation matrix $R(i, j, \theta)$
Rotate the matrices.

$$A'_k = R(i, j, \theta)^T A_k R(i, j, \theta)$$

Accumulate transformations

$$V' = VR(i, j, \theta)$$

endfor
endfor
Get sum of squares of off diagonal elements for all matrices.
Check the stopping criteria
Go to next sweep if previous step is fulfilled.

3.2 Monitoring stopping criteria

The stopping criteria for Jacobi Algorithm on several matrices will be discussed basing on the sum of squares of off diagonal elements. Since in each rotation, the sum of squares of off diagonal elements decreases; it reaches a time when the calculated angle θ goes to zero. At this point, there will be no more transformations, and this means that the sum of squares of off diagonal elements becomes invariant to transformation. We stop the process when this stage is reached. When the algorithm stops $F = v_{ij}(v_{jj} - v_{ii})$ will be small which means that vector v_{ij} of off diagonal elements is orthogonal to the vector $(v_{jj} - v_{ii})$ of difference of diagonal elements.

3.3 How to choose an element

The choice of which element is to be annihilated first is based on the cyclic Jacobi. Here we use the most popular ordering scheme, the row-wise scheme.

Chapter 4

Jacobi Algorithm to compute SVD

Using the Jacobi algorithm on several matrices in chapter 3, whose purpose is to diagonalize several dense symmetric matrices simultaneously, we can also use the same algorithm to compute singular values and singular vectors of several dense matrices.

4.1 Formulation

From the *Singular Value Decomposition* (SVD) point of view, the SVD of a real non symmetric matrix $\mathbf{A}$ is a factorization,

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$$

where
$U \in \Re^{m\times m}$ is orthogonal,
$V \in \Re^{n\times n}$ is orthogonal, and
$\Sigma \in \Re^{m\times n}$ is diagonal.

It is assumed that the diagonal entries σ_j (the singular values) of Σ are non negative and are in non increasing order, that is, $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_r \geq 0$, where $r = \min(m, n)$. The columns of $\mathbf{U}$ and $\mathbf{V}$ are the left and right singular vectors of $\mathbf{A}$ respectively.

The following algorithm is the general Jacobi algorithm which will be used to find the singular values and singular vectors for real $m\times n$, p matrices $X_1, X_2, X_3, \ldots, X_p$, with $m>n$. Here we assume that the difference between m and n is not so large.

ALGORITHM J3

Given the matrices $X_1, X_2, X_3, \ldots, X_p$, *with* $m>n$
Get $A_k = X_k^T X_k$
Set $V=I$
Find the angle θ
Rotate the matrices with $R(i, j\theta)$, *(Jacobi rotation)*
X *is transformed from the right.*

$$V = VR(\theta)$$

Get $C_k = X_k X_k^T$
Set $U=I$
Find the angle θ
Rotate the matrices with $R(i, j\theta)$, *(Jacobi rotation)*

X *is transformed from the left.*

$$U = UR(\theta)$$

The approximate SVD of X_k *is obtained as*

$$\Sigma_k = U^T X_k V \tag{4.1}$$

V *and* U *are the right and left singular vectors respectively.*

Now, to use Jacobi algorithm, on real, p, $m \times n$ matrices $X_1, X_2, X_3, \ldots, X_p$, with $m >> n$, we use the same method as above but with slight modifications, because m is often very much larger than n.

ALGORITHM J4

Get $A_k = X_k^T X_k$
Set $V=I$
Apply Jacobi transformation $R(\theta)$ *in coordinates* i, j *of* A_k *to get the right orthogonal columns of* X_k, *i.e (V)*
X *is transformed from the right.*

$$V = VR(\theta)$$

Get $G = [X_1, X_2, X_3, \ldots, X_p]$

Get Q *from* **QR** *factorization of* G *that is*

$$[Q, r] = qr(G, 0)$$

Let $Y_k = Q^T X_k$ *this reduce the dimension of the matrix.*
Get $C_k = Y_k Y_k^T$
Set $U{=}I$
Apply Jacobi transformation $R(\theta)$ *in coordinates* i, j *of the symmetric matrix* C_k *to get the left orthogonal columns of* X_k*, i.e (U)*

X *is transformed from the left*

$$U = UR(\theta)$$

The approximate SVD of X_k *is obtained by*

$$\Sigma_k = U^T Q^T X_k V \tag{4.2}$$

V *is the right singular vector and* U *the left singular vector.*

The two algorithms above are almost the same, since they all find the singular values and singular vectors for non symmetric dence matrices. The only difference is that the later one is able encounter the situation when the difference between m and n is so large by first performing the **QR** factorization.

4.2 Convergence and Stopping criteria

To study the convergence, we use the same way we did in the Jacobi algorithm for several matrices, that we look on the sum of squares of off diagonal elements for each sweep. We realize that convergence is a bit defferent from previous. Here sum of squares of off diagonal elements converge very slowly in the beginning, and then they remains constant. This means that the angle θ that transform the matrices is very small that it can not reduce the elements any more. When this point is reached we set the stop criteria. So the process will stop as soon as there is no more changes in the sum of squares of off diagonal elements, that is the absolute value of angle θ is very small to a given tolerance.

Chapter 5

Numerical Tests on the Algorithm

Using Jacobi Algorithm on one matrix, several matrices and on SVD, we perform several convergence tests basing on Frobenius norm. We first test with standard matrices and then test with matrices obtained from determination of chemical composition of a sample mixture, where by light of varying wave length is shone on to the sample and the wave length of the reflected light is recorded. In this way the spectrum of the sample is obtained. The results of this experiment are stored in a matrix with m rows and n columns, where n is the number of spectra(recorded at different locations and angles) and m is the number of points at which each spectrum is sampled. Usually $m >> n$.

5.1 Convergence test with one symmetric matrix

Here we test Jacobi algorithm using matrices like Hilbert matrices and Square membrane matrices, the convergence is quadratic and is reached at about the fifth sweep.

5.1.1 Testing with Hilbert Matrices

Hilbert matrices are symmetric matrices with elements
$H_{ij} = (i + j - 1)^{-1}$ for $i, j = 1, 2, \ldots n$
Testing with these matrices in their original form, that is without perturbing them, the convergence is fast, as in figure 5.1.

5.1.2 Testing with Square membrane Matrices

Square membrane matrices are large matrices with some **double** eigenvalues. We test the algorithm using one matrix of this class and the convergence was as expected in figure 5.2

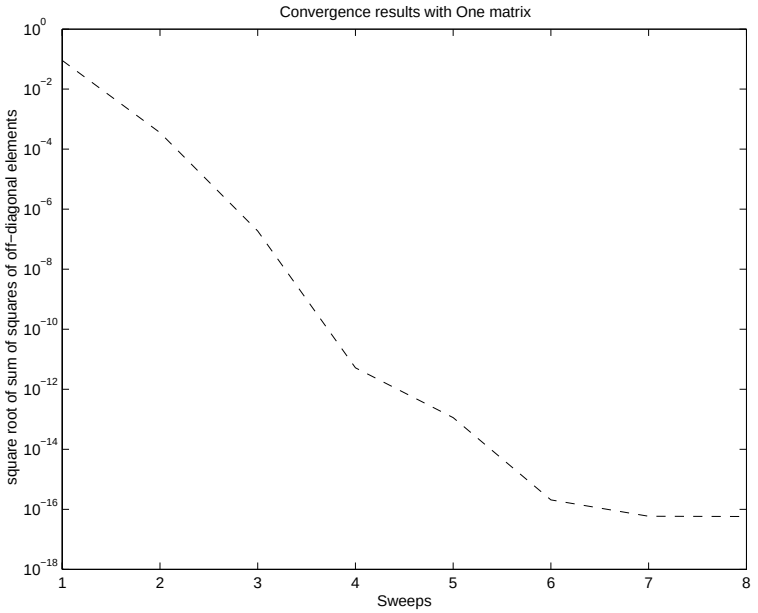


Figure 5.1: Convergence plot using Hilbert matrix. The transformations was stopped when the sum of squares of off diagonal elements becomes very small, as seen, the convergence is fast.

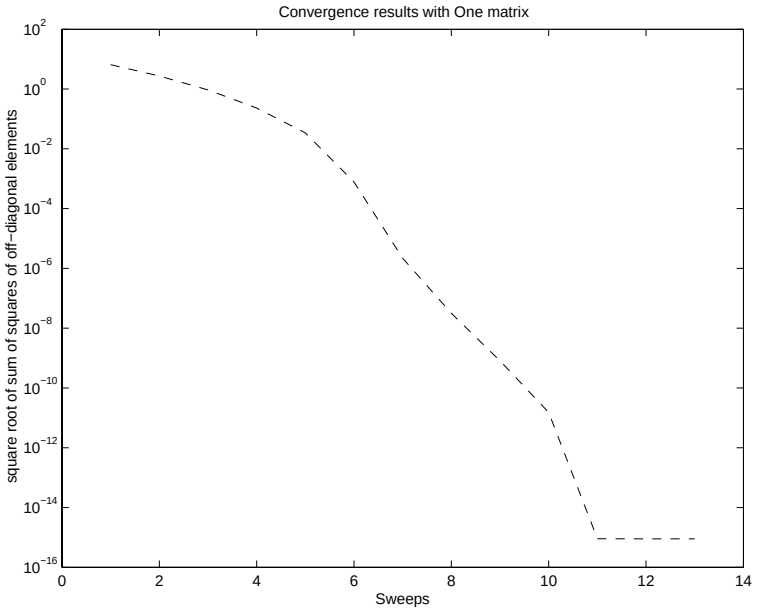


Figure 5.2: Convergence plot with One Square membrane matrix. The sum of squares of off diagonal elements decreases reasonably in each sweep. We stop transformations when they get small to 10^{-16}.

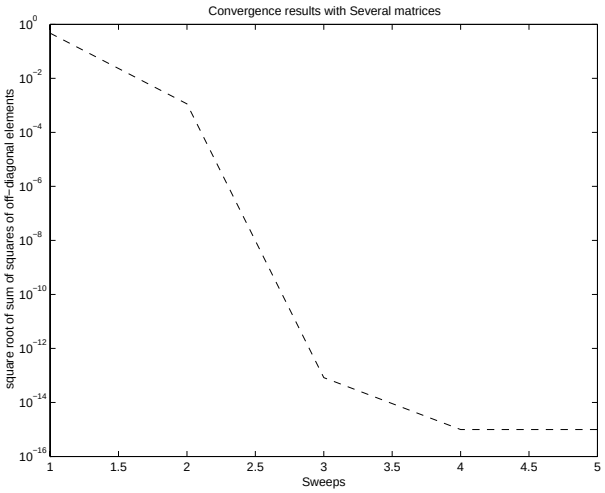


Figure 5.3: Convergence plot using several commuting matrices. Sum of squares of off diagonal elements decreases very fast from the beginning. This is because of the commuting property of the matrices. We stop the transformations when the Sum of squares of off diagonal elements were reasonable small.

5.2 Testing with several Matrices

We test Jacobi algorithm using several matrices that commute each other, that is they have the same eigenvectors. Then we perturb them and study the convergence.

5.2.1 With several commuting Matrices

Here Algorithm J2 is tested using commuting matrices created in the following way:-

1. *Choose n, size of columns of the matrices you want to have.*
2. *Create normally distributed random numbers of n dimension. e.g L=randn(n).*
3. *Do the QR factorization of L that is* $[Q, r] = qr(L)$
4. *Depending on the number of matrices you want to have, get real eigenvalues in column vectors* $D_k, \quad k = 1, 2, 3...p$
5. *Create the matrices from* $A_k = QD_kQ^T$

After generating our matrices above, we run the Algorithm J2. We noticed that the convergence was very fast, as seen in figure 5.3

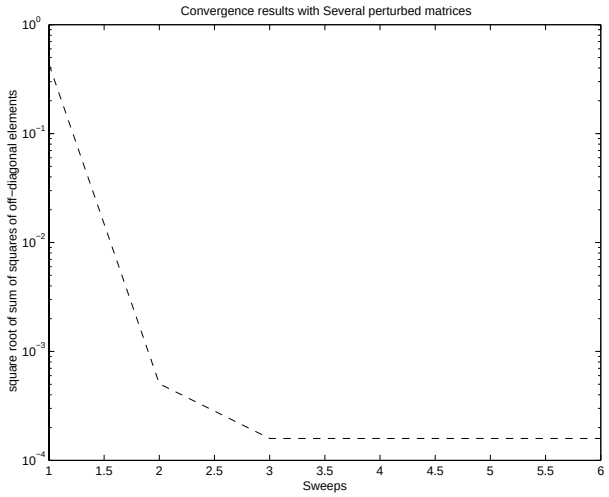


Figure 5.4: Convergence using several perturbed commuting matrices. The matrices are nearly commuting. The size of pertubation is around 1e-05, and as seen the sum of squares goes down reasonably fast in the beginning.

5.2.2 With several perturbed commuting Matrices

We perturb the commuting matrices symmetrically and randomly with different values of δ for instance δ can be $10^{-15}, 10^{-5}, 10^{-3} \ldots$, then transform them (run Algorithm J2 on them). Our intention was to see what will happen if commutivity is disturbed a bit, can the matrices still be diagonalized? What are the effects of perturbations on the convergence speed?

The steps involved to perturb the matrices were as follows

1. *Having* $A_k = QD_kQ^T$
2. *With different values of* δ*, generate normally distributed random numbers* $E_k = \delta * randn(n)$
3. *Then perturb the matrices as follows* $A_k = QD_kQ^T + E_k$
4. *With* A_k *call algorithm J2*

Results in figure 5.4 show that the speed of convergence decreases with the increase in pertubations size. So when the perturbations are large we can not diagonalize the matrices at all. This can be seen in figure 5.5

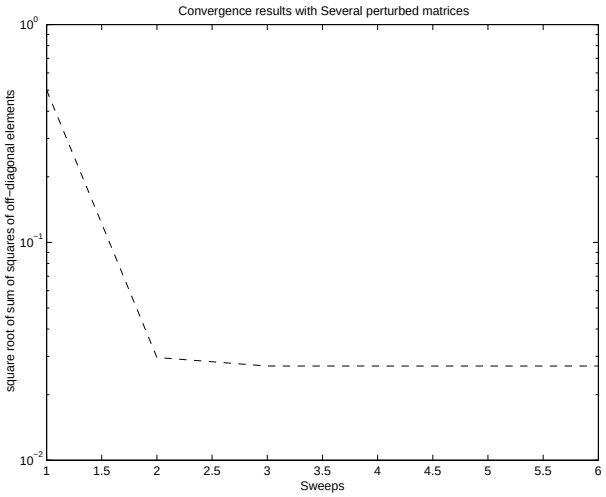


Figure 5.5: Convergence plot using several perturbed commuting matrices. The size of pertubation is around 1e-02. The effect of large perturbations is seen very early, and the off diagonal elements did not decrease, since the matrices are not *nearly commuting*.

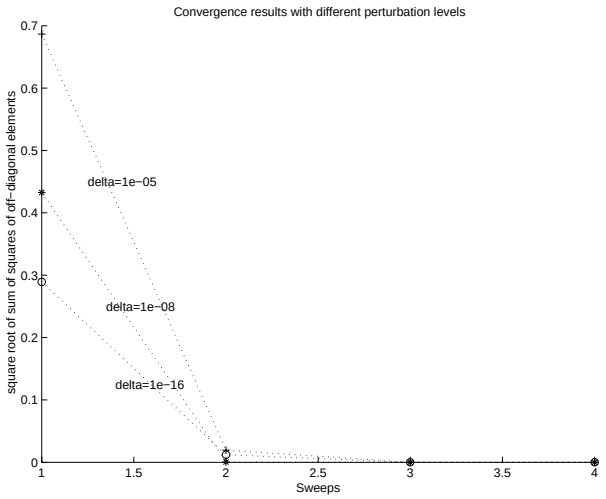


Figure 5.6: Convergence plot showing the variation of convergence speed with δ . As seen the convergence speed decreases with increasing δ.

5.3 Testing with Test Matrices for SVD

The data matrices as explained in the beginning of the chapter were tested as follows:-

5.3.1 With single Test Matrix for SVD

First, we test these matrices using only one matrix at a time. Here we are interested in the singular value decomposition of A. So we just make the matrix symmetric and do the diagonalization basing on the following steps.

1. *Input the matrix* X, X *is* $m \times n$ $m >> n$
2. *Make* $A = X^T X$ *an* $n \times n$ *symmetric matrix.*
3. *With A, call algorithm J2,* *with* V *as right singular vector of* X

   Here X *is transformed from the right*

   To transform X *from left*
4. *Input the matrix* X, A *is* $m \times n$ $m >> n$
5. *Make QR factorization, that is* $[Q, r] = qr(X, 0)$
6. *Then* $X = Q^T X$
7. *Make* $C = XX^T$ *an* $n \times n$ *symmetric matrix.*
8. *With C, call algorithm J2,* *with* U *as left singular vector of* X

The approximate SVD of X *is obtained by*

$$\Sigma = U^T Q^T X V \tag{5.1}$$

Convergence was as expected for single matrix, and is shown in figure 5.7

5.3.2 With several Test Matrices for SVD

The tests on our algorithm continue with more than one matrices, the folowing steps were used. Note that for the SVD case the matrices have to be transformed from both directions, so as to get left and right singular vectors.
Given the matrices $X_1, X_2, X_3, \ldots, X_p$, *with* $m >> n$

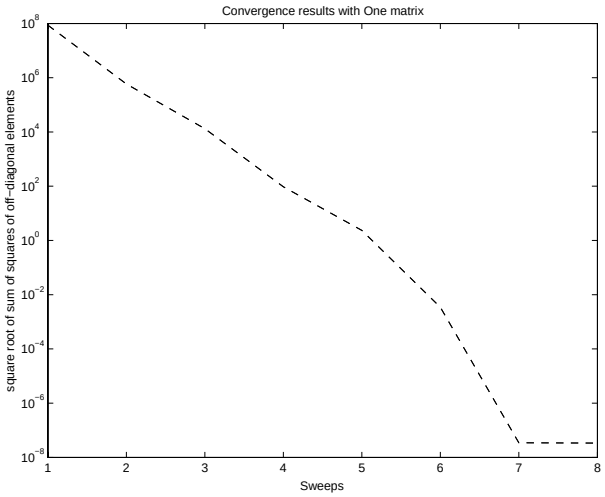


Figure 5.7: Convergence plot using Single Data matrix. The sum of squres of off diagonal elements converge slowly in the beginning and later converge very fast. We stop the transformations when the difference between the sum of squares became reasonably small.

1. *Get* $A_k = X_k^T X_k$.

2. *Having* $A_1, A_2, A_3 \ldots A_p$, *call Algorithm J2. colums of* V *are the right singular vectors of* X_k
   Here $X_1, X_2, X_3, \ldots, X_p$ *are transformed from the right.*

   To transform $X_1, X_2, X_3, \ldots, X_p$ *from left*

3. *Get* $G = [X_1, X_2, X_3, \ldots, X_p]$

4. *Get* Q *from* **QR** *factorization of* G *that is*

   $[Q, r] = qr(G, 0)$

5. *Let* $Y_k = Q^T X_k$

6. *Get* $C_k = Y_k Y_k^T$

7. *Having* C_k, *call Algorithm J2.* k*=1,2 3...*p
   colums of U *are the left singular vectors of* X_k
   The matrices $X_1, X_2, X_3, \ldots, X_p$ *are transformed from the left.*

The approximate SVD of X_k *is obtained by*

$$\Sigma_k = U^T Q^T X_k V \tag{5.2}$$

The results were matrices $D_k s$ which are nearly diagonal which contains squared singular values of $X_k s$.

5.4 Comparison Tests

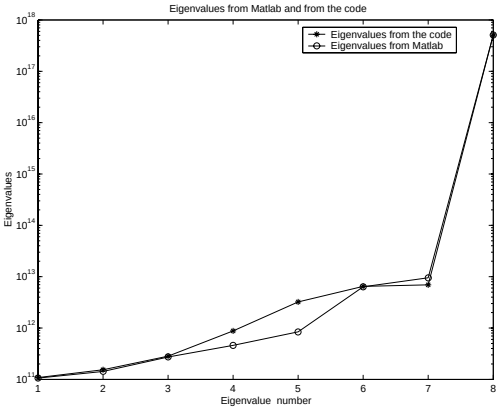
We perform some tests to compare our algorithm with the Matlab functions that compute eigenvalues and singular values of matrices. Though in Matlab these functions compute the values of one matrix at a time.

To start with, we simultaneously diagonalize 5 matrices using our algorithm, later using Matlab functions we compute the eigenvalues of these 5 matrices separately. We then plot the eigenvalues obtained from both ways on the same plot for each matrix. The results arre shown in figure 5.8.

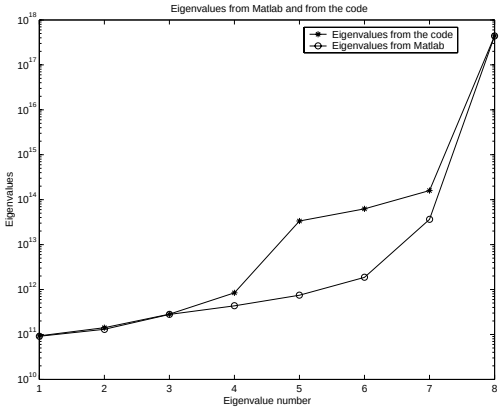
To compare the singular values from our algorithm and Matlab functions, we compute singular values of 5 matrices simultaneously and plot them on the same plot with those computed from Matlab. Figure 5.9 shows the plots.

We tried to look on the singular vectors computed using Matlab functions and from our code. The following matrix is the product of matrix of singular vector Vm from Matlab and matrix of singular vector Vc from our code for the first four columns. As can be seen the columns of these matrices seems to be nearly orthogonal, as their product is nearly identity matrix. However, due to the fact that the matrix Vc is a common matrix,(same for all 5 matrices), this gives us the angle between the singular vectors of one of the matrices computed by Matlab and the common singular vectors estimated by our code. We see that the direction of the leading singular vectors is very close to the leading common vectors. The following to vectors are also rather close, but there is no reason to assume that the matrices have more than 3 singular directions in common.This gives us a proof that our algorithm gives the expected results for matrices which are not commuting.

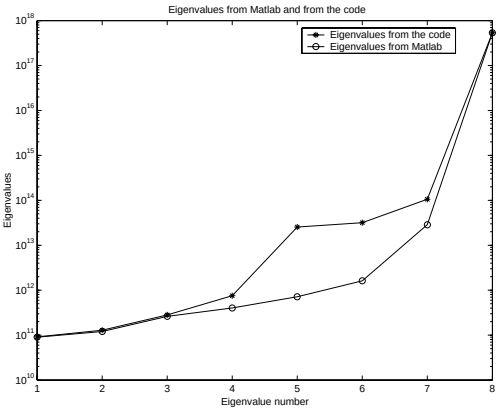
$$\mathbf{V'_m V_c} = \begin{bmatrix} -1.0000e+00 & +1.8992e-05 & +1.1744e-06 & +2.2473e-05 \\ -2.2212e-05 & -9.9581e-01 & -2.6732e-02 & -8.1958e-02 \\ -9.7989e-07 & +5.3022e-02 & -9.0622e-01 & -3.8857e-01 \\ -2.6918e-04 & -5.8599e-02 & -2.0445e-01 & +6.4174e-01 \\ +7.5146e-04 & -3.2616e-02 & -9.1385e-02 & +3.3494e-01 \\ +3.4483e-05 & -9.7780e-03 & -3.4060e-01 & +4.6728e-01 \\ +2.3545e-04 & +2.3448e-02 & +1.0283e-01 & -3.1501e-01 \\ -2.3818e-05 & -3.6273e-02 & +2.6329e-02 & -4.6572e-01 \end{bmatrix} \tag{5.3}$$



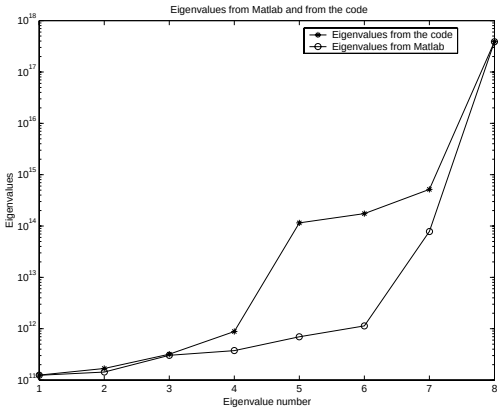
(a) First matrix. .



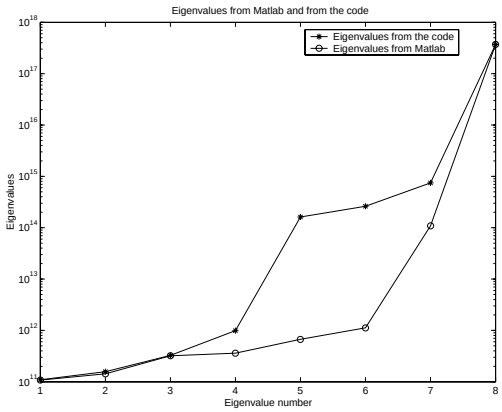
(b) Second matrix.



(c) Third matrix.

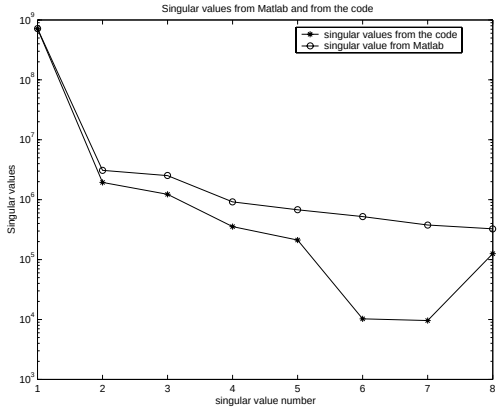


(d) Fourth matrix.

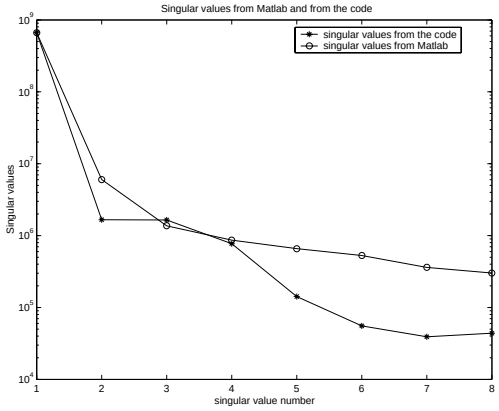


(e) Fifth matrix.

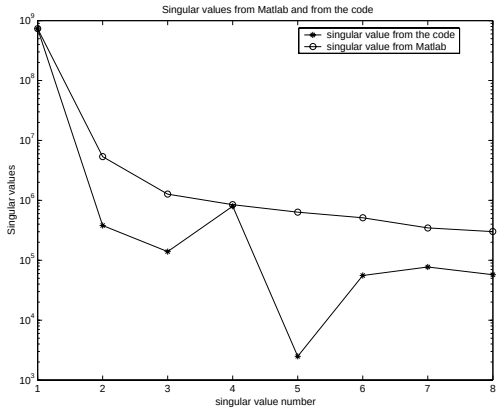
Figure 5.8: The plots for the Eigenvalues from matlab and from the code. The eigenvalues are in increasing order.



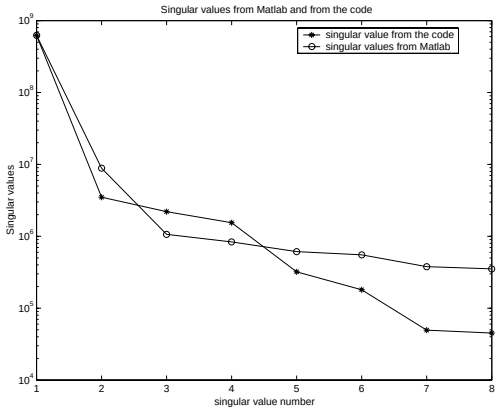
(a) First matrix. .



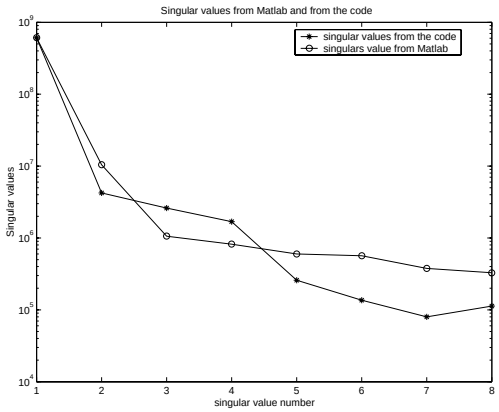
(b) Second matrix.



(c) Third matrix.



(d) Fourth matrix.



(e) Fifth matrix.

Figure 5.9: The plots for the singular values from matlab and from the code.The singular values are in decreasing order.

Chapter 6

Conclusion

In this thesis we tried to create an algorithm that diagonalize several symmetric matrices simultaneously. For the convergence, we concentrate mainly on looking on the sum of squares of off diagonal elements of all the matrices if they get minimized or not. We learn that the degree of convergence depended very much on the commutivity property of the matrices. We found that when the matrices are commuting or nearly commuting, we get full diagonalization and hence minimization of sum of squares of off diagonal elements. In this case the eigenvalues obtained from the algorithm were the same as those computed from Matlab functions. Figure 6.1 plots the eigenvalues obtained when this property is fulfilled. The convergence was reasonably fast. With this property, we plot the singular values obtained from the algorithm and from Matlab functions. The singular values were the same in both as in figure 6.2. Convergence was fast.

When matrices are different, minimization is very slightly, this means that diagonalization is not full (partial diagonalization). In this case it is not possible to get all eigenvalues or singular values exact. We got the first eigenvalue and singular value of each matrix exactly as with Matlab functions. The rest were a bit diferent as seen in figures 5.8 and 5.9.

The minimization of sum of squares of off diagonal elements, depends on the angle θ which was calculated using both off diagonal elements and diagonal elements. It reaches a time that no matter how many rotations you perform, there is no further decrease in those off diagonal elements. This means that the angle θ is zero or is very small and that $\mathbf{A}'$ is diagonal or can not be diagonalized any further. At this stage we set the stopping criteria and assume that our matrix is diagonal or nearly diagonal. It is diagonal if all the off diagonal elements are very small to those in the diagonal and hence the diagonal elements of $\mathbf{A}'$ are the eigenvalues of $\mathbf{A}$ or the diagonal elements of $\mathbf{\Sigma}$ are the singular values of $\mathbf{X}$.

So from the results we got from tests with matrices for SVD, and especially from the matrix 5.3, where the first three columns gives diagonal elements which are 1,0.9958 and 0.90622, we are able to say that the sample mixture contained three species, that is the first non zero singular values.

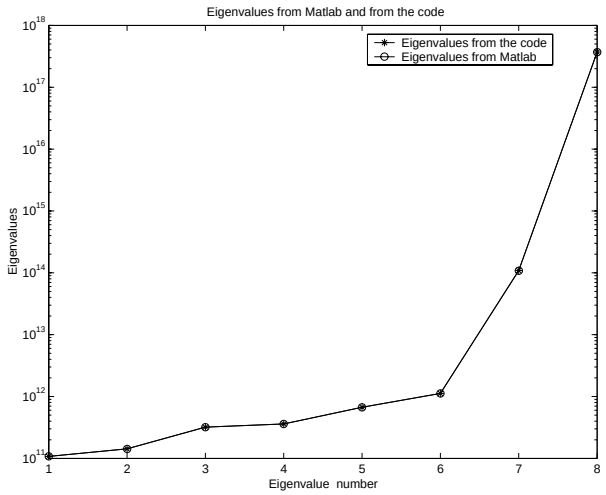


Figure 6.1: The figure shows eigenvalues of commuting matrices computed from matlab and from the code. The eigenvalues are in increasing order.

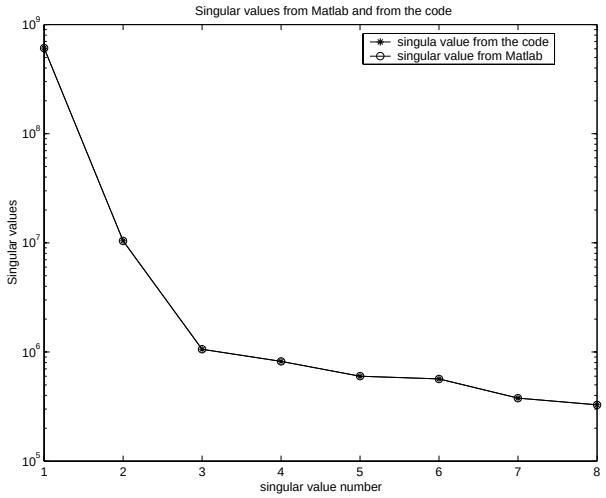


Figure 6.2: The figure shows singular values of commuting matrices computed from matlab and from the code. The eigenvalues are in increasing order.

Bibliography

[1] Beresford N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall Series In Computational Mathematics, ©1980 chapter 6 and 9.

[2] Gremund Dahlquist, and Åke Björck, *Numerical Mathematics*, Volume 2, manuscript (August 1998)

[3] Lloyd N.Trefethen and David Bau, III *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics(SIAM) ©1997.

[4] G. H. Golub and C.F. Van Loan, *Matrix computations*, Baltimore,MD:The John Hopkins Univ.Press, ©1989.

[5] James. W. Demmel, *Applied Numerical Linear Algebra*, Society for Industrial and Applied Mathematics(SIAM) ©1997.

[6] G. W.Stewart, *Matrix algorithms,Volume 1: Basic Decompositions* Society for Industrial and Applied Mathematics (SIAM) ©1998.

[7] Michael T. Heath, *Scientific Computing. An introductory Survey*, McGraw-Hill Series in Computer Science ©1997.