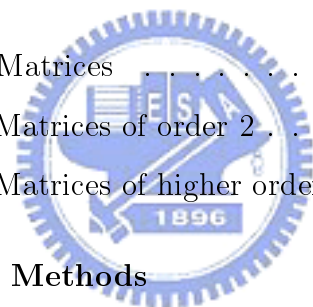


# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Eigenvalue Problem in Quantum Chemistry . . . . .	1
1.2	General Eigenvalue Problem . . . . .	3
1.2.1	Rayleigh Quotient . . . . .	5
1.2.2	Projection . . . . .	5
1.2.3	Orthogonalization Processes . . . . .	6
1.2.4	Gerschgorin's Theorems . . . . .	8
<b>2</b>	<b>Direct Methods</b>	<b>9</b>
2.1	Diagonalization of Matrices . . . . .	9
2.1.1	Symmetric Matrices of order 2 . . . . .	9
2.1.2	Symmetric Matrices of higher order . . . . .	10
<b>3</b>	<b>Overview of Iterative Methods</b>	<b>12</b>
3.1	Main Idea . . . . .	12
3.2	Iterative Methods . . . . .	14
3.2.1	Lanczos Method . . . . .	14
3.2.2	Davidson Method . . . . .	14
3.2.3	Jacobi-Davidson Method . . . . .	15
3.3	Other Notations . . . . .	15
3.3.1	Constructing Initial Vector $\mathbf{g}_1$ . . . . .	15
3.3.2	Restart Strategies . . . . .	16
<b>4</b>	<b>Loss of Orthogonality</b>	<b>17</b>
4.1	An Easy Example . . . . .	17
4.2	Examples using Davidson Method . . . . .	18



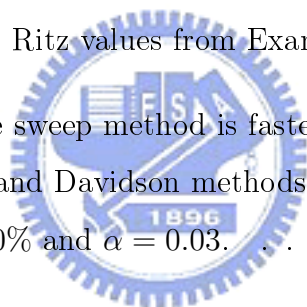
---

<b>5</b>	<b>The Sweep Method</b>	<b>23</b>
5.1	Jacobi-sweep Method . . . . .	23
5.2	Main Idea . . . . .	24
5.3	Results . . . . .	24
5.4	Conclusions . . . . .	28
<b>A</b>	<b>Program Codes</b>	<b>31</b>
A.1	Main Program . . . . .	31
A.2	Davidson Preconditioner . . . . .	36
A.3	Jacobi-Davidson Preconditioner . . . . .	36
A.4	Sweep Preconditioner . . . . .	37
<b>B</b>	<b>Numerical Results for Small Matrices</b>	<b>43</b>
<b>C</b>	<b>Numerical Results for Large Matrices</b>	<b>46</b>



# List of Figures

1.1	Potential energy curve for the $H_2$ molecule. The bond corresponds to the minimum in the curve. . . . .	3
4.1	Location of the largest 10 eigenvalues. . . . .	19
4.2	Residual values and Ritz values from Example 1. . . . .	19
4.3	Overlap matrix $\mathbf{O}_{30}$ from Example 1. . . . .	20
4.4	Graphical presentation of $\hat{\mathbf{A}}$ . . . . .	20
4.5	Location of all eigenvalues of $\tilde{\mathbf{A}}$ . . . . .	21
4.6	Residual values and Ritz values from Example 2. . . . .	21
4.7	Residual values and Ritz values from Example 3. . . . .	22
5.1	Convergence for the sweep method is faster than for Davidson method. . . . .	25
5.2	$\bar{\mathbf{A}}$ using the sweep and Davidson methods. . . . .	26
5.3	A sample for $\rho = 30\%$ and $\alpha = 0.03$ . . . . .	28



# Chapter 1

## Introduction

### 1.1 Eigenvalue Problem in Quantum Chemistry

For very small objects like molecules, we can not understand their behavior by Newton's laws. The Schrödinger equation is used to predict their properties and behavior. Let  $\mathbf{x} = (x_1, x_2, \dots, x_p)$ . The Schrödinger equation has the form of

$$\left[ -\frac{\hbar^2}{2m} \left( \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \dots + \frac{\partial^2}{\partial x_p^2} \right) + V(\mathbf{x}) \right] \Psi(\mathbf{x}) = E\Psi(\mathbf{x}), \quad (1.1)$$

where  $V(\mathbf{x})$  is an external potential. The wave function  $\Psi(\mathbf{x})$  can be interpreted as the probability of finding molecules at a given position at a given time.  $E$  denotes the energy characterizing some stable state of a molecule; the structure of the molecules (e.g., bond lengths) can be determined by means of it.

Nevertheless, it is difficult to solve this differential equation analytically, so we need another way to achieve it. Define the operator

$$\hat{A} = -\frac{\hbar^2}{2m} \left( \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \dots + \frac{\partial^2}{\partial x_p^2} \right) + V(\mathbf{x}). \quad (1.2)$$

Hence the Schrödinger equation can be reduced to

$$\hat{A}\Psi(\mathbf{x}) = E\Psi(\mathbf{x}). \quad (1.3)$$

Now, it has been transformed into an eigenvalue problem, where  $E$  and  $\Psi(\mathbf{x})$  can be considered as the eigenvalue and eigenfunction of  $\hat{A}$ , respectively. The next step is to change the operator  $\hat{A}$  to a matrix form.

Let  $\Psi \in \mathcal{F}$ , where  $\mathcal{F}$  is a Hilbert space. We choose an orthonormal basis in  $\mathcal{F}$  :  $\{g_i(\mathbf{x}) : i = 1, 2, \dots\}$ , then  $\Psi(\mathbf{x}) = \sum_{i=1}^{\infty} c_i g_i$ , where  $c_i = \int f_i^* \Psi d\mathbf{x}$ . In practice,

it is often possible to replace the complete basis set by some finite set of functions :  $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_n(\mathbf{x})$ . After replacing  $\Psi(\mathbf{x})$  in Equation 1.3 by  $\sum_{i=1}^n c_i f_i(\mathbf{x})$ , we have

$$\hat{A} \left( \sum_{i=1}^n c_i f_i(\mathbf{x}) \right) = E \left( \sum_{i=1}^n c_i f_i(\mathbf{x}) \right). \quad (1.4)$$

For  $i = 1, 2, \dots, n$ , premultiply  $f_i(\mathbf{x})$  and integrate both sides in Equation 1.4:

$$\sum_{j=1}^n c_j \int f_i^* \hat{A} f_j d\mathbf{x} = E \left( \sum_{j=1}^n c_j \int f_i^* f_j d\mathbf{x} \right). \quad (1.5)$$

By the fact  $\int f_i(\mathbf{x})^* f_j(\mathbf{x}) d\mathbf{x} = \delta_{i,j}$ ,

$$\sum_{j=1}^n c_j \int f_i^* \hat{A} f_j d\mathbf{x} = E c_i. \quad (1.6)$$

Let the matrix  $\mathbf{A} = [\mathbf{a}_{i,j}]$ , where

$$\mathbf{a}_{i,j} = \int f_i^* \hat{A} f_j d\mathbf{x}, \quad (1.7)$$

where  $f_i, f_j \in \mathcal{F}$ . From Equation 1.6, we have

$$\mathbf{A} \mathbf{c} = E \mathbf{c}, \quad (1.8)$$

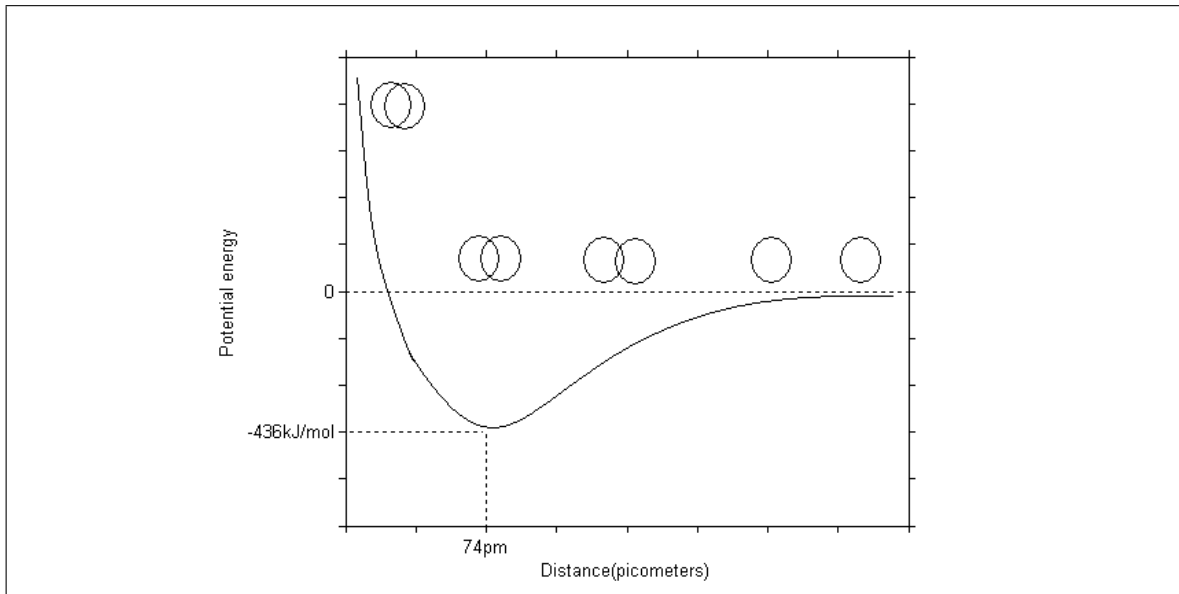
where  $\mathbf{c} = [c_1, c_2, \dots, c_n]^T$ . In general,  $\mathbf{A}$  is a symmetric matrix since  $\hat{A}$  is a symmetric operator, i.e.

$$\int f^* \hat{A} g d\mathbf{x} = \int g (\hat{A} f)^* d\mathbf{x} \quad (1.9)$$

where  $f, g \in \mathcal{F}$ . The form of the problem has been changed from a differential equation to a linear system

$$(\mathbf{A} - E\mathbf{I}) \mathbf{c} = \mathbf{0}. \quad (1.10)$$

You can find the structure of molecules using the eigenvalues of  $\mathbf{A}$ . In Figure 1.1, there is no attraction between two hydrogen atoms when the distance between them is large. Once they get closer, their potential energy is lowered. When the energy is in minimum, it corresponds to a bond connecting these two atoms. The value of energy they possess in the minimum can be interpreted as the minimal eigenvalue of  $\mathbf{A}$ . This is the physical meaning for the minimum eigenvalue. In general, calculating the eigenvalues of  $\mathbf{A}$ , allows for finding the bond lengths of the molecule by other procedures. Then you will understand the structure of molecules.



**Figure 1.1** Potential energy curve for the  $H_2$  molecule. The bond corresponds to the minimum in the curve.

## 1.2 General Eigenvalue Problem

The general eigenvalue problem is

$$\mathbf{A}\mathbf{u} = \lambda\mathbf{u}, \quad (1.11)$$

where  $\lambda$  is called an eigenvalue of the matrix  $\mathbf{A}$  of order  $n$  and  $\mathbf{u}$  is the eigenvector corresponding to  $\lambda$ . You can consider the eigenvalue problem as a linear system since it can be modified as

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{u} = \mathbf{0}. \quad (1.12)$$

The linear equation has a nonzero solution  $\mathbf{u}$  when  $(\mathbf{A} - \lambda\mathbf{I})$  is singular, i.e. the determinant of  $(\mathbf{A} - \lambda\mathbf{I})$  is equal to zero, i.e.,  $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$ . We derive a characteristic polynomial

$$\lambda^n + c_{n-1}\lambda^{n-1} + \cdots + c_1\lambda + c_0 = 0. \quad (1.13)$$

from  $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$ . There are  $n$  roots  $\lambda_1, \lambda_2, \dots, \lambda_n$  of the characteristic equation, and they are the eigenvalues exactly. Plug  $\lambda$  into Equation 1.12 to each eigenvalue, you can find all the eigenvectors  $\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_n$  respect to them.

Let us assume that,

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n.$$

We name the set of these eigenvalues the spectrum of  $\mathbf{A}$  and denote it by  $\sigma(\mathbf{A})$ .

We can generalize the eigenvalue problems into an integral form

$$\int_a^b G(x, y)f(x)dx = \lambda f(y), \quad (1.14)$$

where  $G(x, y)$  is the kernel of the integral operator. Then we do not just have the problem in a matrix form, but rather a continuous case.

There are many scientific fields that extensively use eigenvalue problems like quantum chemistry. After abstracting these problems, they may look similar. The matrices involved in the eigenvalue problems can be divided into two types, Hermitian matrices and non-Hermitian matrices. In this thesis, we consider only Hermitian matrices. Further, Hermitian matrices can be reduced to symmetric ones since we only use real-valued arithmetics.

The goal in eigenvalue problems is to find the eigenpairs of a given matrix, but requirements for the eigenvalues in different problems are not the same. In some problems you have to find all the eigenvalues of a small matrix, and others to look for the extreme eigenvalues of a large one. The two essential elements in calculating the eigenpairs of a large matrix are the computer memory requirements and accuracy. The efficiency is also important since we use large matrices (of order  $10^6$  or even larger). Fortunately, usually the matrices are diagonally dominant and sparse. On average, maybe only 10% or fewer of the elements of a large matrix are nonzeros.

There exist ways of reducing the memory requirements. The main cost of calculations using iterative methods is the constant matrix-vector product operations. The accuracy is hard to check because we can not compute the exact eigenpairs of a large matrix directly, but there is still some bounds for eigenvalues. Once the problems with accuracy are eliminated, we should concentrate on improving the efficiency of our algorithms.

A fact for symmetric matrices is that all the eigenvalue and eigenvectors of them are real. Particularly,

$$\mathbf{x}^T \mathbf{A} \mathbf{x} \in \mathbf{R} \text{ for all } \mathbf{x} \in \mathbf{R}^n.$$

This is a very important property since it allows us for employing the Rayleigh quotient techniques to study our problems.

### 1.2.1 Rayleigh Quotient

Let  $\mathbf{x}$  be a nonzero vector. The Rayleigh quotient is defined as

$$\phi(\mathbf{x}) = \frac{\mathbf{x}^T \mathbf{A} \mathbf{x}}{\mathbf{x}^T \mathbf{x}} \quad (1.15)$$

The range of  $\phi(\mathbf{x})$  is called the field of values of  $\mathbf{A}$ , and it is the interval  $[\lambda_1, \lambda_n]$ . That means, the minimum value of  $\phi(\mathbf{x})$  is obtained if and only if  $\mathbf{x}$  is the eigenvector corresponding to  $\lambda_1$ , and the maximum value of  $\phi(\mathbf{x})$  is obtained if and only if  $\mathbf{x}$  is the eigenvector corresponding to  $\lambda_n$ .

Since  $\lambda_1$  and  $\lambda_n$  are finite numbers, and so  $\phi(\mathbf{x})$  is bounded. The Rayleigh quotient is important both for theoretical and practical purposes. Sometimes, it is convenient to restrict the above definitions to a unit sphere in  $\mathbf{R}^n$ :

$$\phi(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} \text{ for } \|\mathbf{x}\| = 1. \quad (1.16)$$

These two different definitions lead to the same properties, and therefore we choose the one which is easier to apply in a given situation.

### 1.2.2 Projection

The orthogonal projection operator  $\mathbf{P}$  on a vector  $\mathbf{x} \in \mathbf{R}^n$  is defined as

$$\mathbf{P}(\mathbf{u}) = \mathbf{x}(\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{u}. \quad (1.17)$$

In the eigenvalue problem considerations, we often need to compute the projection of  $\mathbf{A} \mathbf{x}$  on  $\mathbf{x}$  for some  $\mathbf{x} \in \mathbf{R}^n$ , i.e.

$$\mathbf{P}(\mathbf{A} \mathbf{x}) = \mathbf{x}(\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{A} \mathbf{x}. \quad (1.18)$$



Since  $(\mathbf{x}^T \mathbf{x})^{-1}$  is a constant, it implies

$$\mathbf{P}(\mathbf{Ax}) = \frac{\mathbf{x}^T \mathbf{Ax}}{\mathbf{x}^T \mathbf{x}} \mathbf{x} = \phi(\mathbf{x}) \mathbf{x}. \quad (1.19)$$

From above, the projection of  $\mathbf{Ax}$  can be represented as the Rayleigh quotient  $\phi(\mathbf{x})$  times  $\mathbf{x}$ . There also exists another projection form defined through matrices. Suppose  $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_m$  are linearly independent vectors that span a  $m$ -dimensional space  $\mathcal{L} \subseteq \mathbf{R}^n$ . Then, a projection of  $\mathbf{u} \in \mathbf{R}^n$  onto  $\mathcal{L}$  can be defined as

$$\mathbf{P}_G(\mathbf{u}) = \mathbf{G}(\mathbf{G}^T \mathbf{G})^{-1} \mathbf{G}^T \mathbf{u}. \quad (1.20)$$

### 1.2.3 Orthogonalization Processes

Suppose there are  $m$  independent normalized vectors  $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_m$  that need to be orthogonalized. The standard algorithm of Gram-Schmidt process is given as below [2, pages 10–12]:

#### Algorithm 1.1 : Gram-Schmidt

1. First, define  $\mathbf{x}_1$  as  $\mathbf{g}_1$ .
2. For  $j = 2, \dots, m$ , do
3. Calculate  $c_{i,j} = \langle \mathbf{g}_j, \mathbf{x}_i \rangle$ , for  $i = 1, 2, \dots, j - 1$ .
4.  $\hat{\mathbf{x}} = \mathbf{g}_j - \sum_{i=1}^{j-1} c_{i,j} \mathbf{x}_i$ .
5. normalize  $\hat{\mathbf{x}}$  into  $\mathbf{x}_j$ .
6. end do

There is another algorithm called Modified Gram-Schmidt(MGS) that has better numerical properties:

#### Algorithm 1.2 : Modified Gram-Schmidt

1. First, define  $\mathbf{x}_1$  as  $\mathbf{g}_1$ .

2. For  $j = 2, \dots, m$ , do
3.  $\hat{\mathbf{x}} = \mathbf{g}_j$
4. For  $i = 1, \dots, j - 1$ , do
5.  $c_{i,j} = \langle \hat{\mathbf{x}}, \mathbf{x}_i \rangle$
6.  $\hat{\mathbf{x}} = \hat{\mathbf{x}} - c_{i,j} \mathbf{x}_i$ .
7. end do
8. normalize  $\hat{\mathbf{x}}$  into  $\mathbf{x}_j$ .
9. end do

You can have better way than MGS if you change the inside for loop (from Step 4 to Step 7) in MGS. First of all, let  $\Gamma = \{1, 2, \dots, j - 1\}$ . Every time, find  $\mathbf{x}_p$  for  $p \in \Gamma$  such that  $\langle \hat{\mathbf{x}}, \mathbf{x}_p \rangle = \max_{i \in \Gamma} \langle \hat{\mathbf{x}}, \mathbf{x}_i \rangle$ , and then do  $\hat{\mathbf{x}} = \hat{\mathbf{x}} - \langle \hat{\mathbf{x}}, \mathbf{x}_p \rangle \mathbf{x}_p$  and  $\Gamma = \Gamma \setminus \{p\}$ . After  $j - 1$  times, all the elements in  $\Gamma$  have been removed, and it means that we have projected  $\hat{\mathbf{x}}$  onto all the vectors  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{j-1}$ . We now show the practical algorithm as follows:

**Algorithm 1.3 : A Change in Modified Gram-Schmidt**

1. First, define  $\mathbf{x}_1$  as  $\mathbf{g}_1$ .
2. For  $j = 2, \dots, m$ , do
3.  $\hat{\mathbf{x}} = \mathbf{g}_j$
4.  $\Gamma = \{1, 2, \dots, j - 1\}$
5. For  $i = 1, \dots, j - 1$ , do
6. Find  $\mathbf{x}_p$  for  $p \in \Gamma$  such that  $\langle \hat{\mathbf{x}}, \mathbf{x}_p \rangle = \max_{i \in \Gamma} \langle \hat{\mathbf{x}}, \mathbf{x}_i \rangle$ .
7. Do  $\hat{\mathbf{x}} = \hat{\mathbf{x}} - \langle \hat{\mathbf{x}}, \mathbf{x}_p \rangle \mathbf{x}_p$  and  $\Gamma = \Gamma \setminus \{p\}$

8. end do
9. normalize  $\hat{\mathbf{x}}$  into  $\mathbf{x}_j$ .
10. end do

Since all the vectors are normalized,  $\langle \hat{\mathbf{x}}, \mathbf{x}_i \rangle$  for  $i = 1, 2, \dots, j - 1$  are values between  $-1$  and  $1$ . If you choose the vector  $\mathbf{x}_i$  for  $i = 1, 2, j - 1$  closest to  $\hat{\mathbf{x}}$ , then projecting  $\hat{\mathbf{x}}$  onto this vector will lead to smallest loss of precision in computer calculations.

### 1.2.4 Gerschgorin's Theorems

Before computing the eigenvalues of matrices, it is helpful to know the range of them. There are theorems on bounds for the eigenvalues as below, and the statements are from [3, page 71–72].

**Theorem 1** *Every eigenvalue of the matrix  $\mathbf{A}$  lies in at least one of the circular discs with centres  $\mathbf{a}_{i,i}$  and radii  $\sum_{j \neq i} |\mathbf{a}_{i,j}|$ .*

**Theorem 2** *If  $s$  of the circular discs of Theorem 1 form a connected domain which is isolated from the other discs, then there are precisely  $s$  eigenvalues of  $\mathbf{A}$  within this connected domain.*

From Theorem 1, if all the circular discs are disconnected, you will be sure that the order of eigenvalues will be the same as the order of the diagonal elements of  $\mathbf{A}$ . Then you know from which diagonal element each of the eigenvalues is originating. The eigenvalues are clustered as the interaction of the discs for larger radii. Theorem 2 says that in such situation, the number of eigenvalues depend on the number of diagonal elements in the connected discs.

These two theorems can let you know if the order of eigenvalues change when you enlarge the radii of discs. It will be more complicated if there are connected discs. That will be a harder problem to handle in such cases.

# Chapter 2

## Direct Methods

I have written programs in FORTRAN 90, using various subroutines provided in LAPACK packages. Using the direct methods to solve problems has the advantage of producing exact answers. The efficiency of these subroutines decreases with increasing requirements for memory, so they are only useful for small matrices. We may apply direct methods for diagonalizing matrices of small order.

### 2.1 Diagonalization of Matrices

The practical way to diagonalize a matrix is quite different from theoretical one. We discuss this for a matrix of order 2 first.

#### 2.1.1 Symmetric Matrices of order 2

In the case of matrices of order 2, it is simple to obtain their complete eigenpairs. Define a  $2 \times 2$  general symmetric matrix  $\mathbf{A}$  as

$$\mathbf{A} = \begin{pmatrix} a & c \\ c & b \end{pmatrix}. \quad (2.1)$$

First compute the roots  $\theta_1$  and  $\theta_2$  of the characteristic equation

$$\theta^2 - (a + b)\theta + (ab - c^2) = 0,$$

and later calculate the eigenvectors by the solutions of  $(\mathbf{A} - \theta_i)\mathbf{x} = 0$  for  $i = 1, 2$ .

### 2.1.2 Symmetric Matrices of higher order

Now, we consider  $\mathbf{A}$  of order  $n > 2$ . In fact, it is hard to solve their characteristic equations like the way we did for matrices of order 2. LAPACK is written in FORTRAN 77 and provides subroutines for solving the most common problems in numerical linear algebra. It provides a subroutine DSYEV, that consist of DSYTRD and DSTEQR, to calculate the eigenpairs.

#### The Subroutine DSYTRD

The subroutine DSYTRD is used to reduce  $\mathbf{A}$  to a tridiagonal form  $\mathbf{T}$  by orthogonal similarity transformations, using Householder's method [4, pages 74–77]. An orthogonal similarity transformation of  $\mathbf{A}$  is given by the elementary symmetric matrix

$$\mathbf{P}^{(r)} = \mathbf{I} - 2\mathbf{w}^{(r)}\{\mathbf{w}^{(r)}\}^T, \quad (2.2)$$

where  $\mathbf{w}^{(r)}$  is a unit vector and defined below.

After doing  $(n-2)$  orthogonal similarity transformations  $\mathbf{A}_{r+1} = \mathbf{P}^{(r)}\mathbf{A}_r\mathbf{P}^{(r)}$  for  $r = 1, 2, \dots, (n-2)$ , where  $\mathbf{A}_1 = \mathbf{A}$ ,  $\mathbf{A}$  will be reduced to a tridiagonal form. Simplistically, let  $\mathbf{x}_r$  be the  $r$ th column of  $\mathbf{A}_r$  that can be represented as

$$\mathbf{x}_r^T = (\hat{\mathbf{x}}^T, x_{r+1}, \mathbf{y}^T) \quad (2.3)$$

where  $\hat{\mathbf{x}}$  is  $r \times 1$ ,  $\mathbf{y}$  is  $\{n - (r + 1)\} \times 1$ .  $\mathbf{P}^{(r)}$  makes sure that the  $(r + 2)$ th to  $n$ th component of  $\mathbf{x}_{r+1}$  will be equal to zero. Corresponding to the form of  $\mathbf{x}_r$ , there is a way to choose  $\mathbf{w}^{(r)}$ , given by

$$\{\mathbf{w}^{(r)}\}^T = (\mathbf{0}, \alpha v_{r+1}^{(r)}, \alpha \mathbf{y}^T) \quad (2.4)$$

where  $v_{r+1}^r = x_{r+1} \pm s$  and  $\alpha = 2^{-1/2}\{s^2 \pm x_{r+1}s\}^{-1/2}$  with  $s = \{(x_{r+1})^2 + \mathbf{y}^T\mathbf{y}\}^{1/2}$ .

#### The Subroutine DSTEQR

The subroutine DSTEQR is used to find the eigenpairs of  $\mathbf{T}$  by employing the implicit QL or QR method [4, pages 85–89]. The LR algorithm is based on the LU decomposi-

tion. Let  $\mathbf{A}_1 = \mathbf{A}$ , and suppose the LU decomposition of  $\mathbf{A}_1$  is

$$\mathbf{A}_1 = \mathbf{L}_1 \mathbf{R}_1 \quad (2.5)$$

where  $\mathbf{L}_1$  is unit lower triangular and  $\mathbf{R}_1$  is an upper triangular matrix. Let

$$\mathbf{A}_2 = \mathbf{R}_1 \mathbf{L}_1 \quad (2.6)$$

By this process, it will lead to

$$\mathbf{A}_s = \mathbf{L}_s \mathbf{R}_s \text{ and } \mathbf{A}_{s+1} = \mathbf{R}_s \mathbf{L}_s. \quad (2.7)$$

$\mathbf{A}_s$  will tend to an upper triangular matrix whose diagonal elements converge to the eigenvalues of  $\mathbf{A}$ .

The QR algorithm from QR decomposition is defined as

$$\mathbf{A}_s = \mathbf{Q}_s \mathbf{R}_s \text{ and } \mathbf{A}_{s+1} = \mathbf{R}_s \mathbf{Q}_s, \quad (2.8)$$

where  $\mathbf{Q}_s$  is a unitary matrix and  $\mathbf{R}_s$  is an upper triangular matrix. Similar to the LR algorithm, you can find the eigenvalues of the sequence  $\mathbf{A}$  by the limit of  $\{\mathbf{A}_s\}$ . Also, columns of the limit converged by the sequence  $\{\mathbf{Q}_1 \cdots \mathbf{Q}_s\}$  represents the eigenvectors of  $\mathbf{A}$ .

You can see the necessity of setting matrices in the computer representation. It is impossible to do so for large matrices because of the limit of the available memory. You may find eigenpairs by these direct methods quickly but they are useless for the matrices of larger order. Because the computer memory is usually limited, we introduce iterative methods in next chapter that allow for determination of extremal eigenpairs without high memory requirements.

# Chapter 3

## Overview of Iterative Methods

In the previous chapter we have discussed some ways to solve the eigenvalue problem for matrices of small order. Now let us turn to the matrices of large order as  $10^6$ . It is a trouble to use this kind of matrices in the computer representation since there is a limit of the available computer memory. Iterative methods need much less memory than direct methods for matrices of large order. Lanczos, Davidson and Jacobi-Davidson methods are examples of iterative methods, which are designed to find extreme eigenvalues. In general Lanczos, Davidson and Jacobi-Davidson methods can find a few extreme eigenvalues at the same time, but here we just look for a single minimum eigenvalue. In Section 3.1, we present the main idea of these methods. In Section 3.2, they will be introduced in more details separately.

### 3.1 Main Idea

We search for the minimum eigenvalue of some matrix  $\mathbf{A}$  using a sequence of subspaces  $\mathcal{K}_m$ . At the beginning,  $\mathcal{K}_1$  contains only a normalized vector  $\mathbf{g}_1$  as the initial vector. At iteration  $m$ , a normalized vector  $\mathbf{g}_{m+1}$  is produced, which is orthogonal to  $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_m$ , and defines  $\mathcal{K}_{m+1} = \text{span}(\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_{m+1})$ .

At iteration  $m$ , define the projector  $\mathbf{P}_m$  onto  $\mathcal{K}_m$  as

$$\mathbf{P}_m = \sum_{i=1}^m \mathbf{g}_i \mathbf{g}_i^T. \quad (3.1)$$

There exists an eigenvector  $\mathbf{y}_m$  corresponding to the minimum eigenvalue  $\lambda_m$  of  $\mathbf{P}_m^T \mathbf{A} \mathbf{P}_m$ .

with restriction to  $\mathcal{K}_m$ . Let  $\mathbf{x}_m = \mathbf{P}_m \mathbf{y}_m$ . If

$$\mathbf{A} \mathbf{x}_m \doteq \lambda_m \mathbf{x}_m, \quad (3.2)$$

then we will say  $(\lambda_m, \mathbf{x}_m)$  is an approximate minimum eigenpair of  $\mathbf{A}$ . By the way,

$$\lambda_m = \min_{\mathbf{w} \in \mathcal{K}_m} \frac{\mathbf{w}^T \mathbf{A} \mathbf{w}}{\mathbf{w}^T \mathbf{w}} \quad (3.3)$$

and so  $(\lambda_m, \mathbf{x}_m)$  is the best approximate eigenpair in  $\mathcal{K}_m$ .

If 3.2 fails, we will define

$$\mathbf{g}_{m+1} = \mathbf{A} \mathbf{x}_m - \lambda_m \mathbf{x}_m. \quad (3.4)$$

Here, Lanczos method does not change the direction of  $\mathbf{g}_{m+1}$ , but Davidson and Jacobi-Davidson methods use different ways to modify it. After normalizing  $\mathbf{g}_{m+1}$  and augmenting  $\mathcal{K}_m$  with  $\mathbf{g}_{m+1}$ , we proceed to next iteration.

It is difficult to execute this idea directly, and so the practical **Algorithm 3.1** is the following:

1. Generate a normalized vector  $\mathbf{g}_1$  as the initial vector.
2. Set  $m = 1$ .
3. Set  $\mathbf{G}_m = [\mathbf{g}_1 \mathbf{g}_2 \dots \mathbf{g}_m]$ .
4. Calculate a small matrix  $\mathbf{A}_m = \mathbf{G}_m^T \mathbf{A} \mathbf{G}_m$ .
5. Calculate the minimum eigenpair  $(\lambda_m, \mathbf{z}_m)$  of  $\mathbf{A}_m$ .
6. Set  $\mathbf{x}_m = \mathbf{G}_m \mathbf{z}_m$ .
7. Calculate the residual vector  $\mathbf{r}_m = (\mathbf{A} - \lambda_m \mathbf{I}) \mathbf{x}_m$ .
  - If  $\|\mathbf{r}_m\| < 10^{-8}$ , exit.
  - If  $\|\mathbf{r}_m\| \geq 10^{-8}$ , continue.

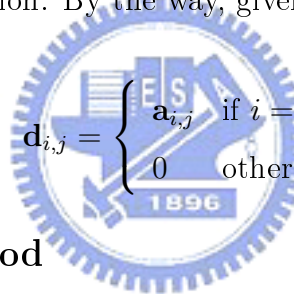


8. Let  $\mathbf{g}_{m+1} = \mathbf{r}_m$  and use Lanczos, Davidson or Jacobi-Davidson strategies to change the direction of  $\mathbf{g}_{m+1}$ .
9. Orthogonalize  $\mathbf{g}_{m+1}$  with respect to  $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_m$  and then normalize it.
10. Set  $m = m + 1$  and go to Step 3.

From Step 3 to Step 7, this algorithm is known as as the Rayleigh-Ritz procedure. It can be proved that it will converge to the same  $\lambda_m$  and  $\mathbf{x}_m$  like the one in the main idea [1, pages 216–217], hence the results are still the same.

## 3.2 Iterative Methods

Lanczos, Davidson and Jacobi-Davidson methods use different strategies in Step 8, and we explain them in this section. By the way, given  $\mathbf{A} = [\mathbf{a}_{i,j}]$  and the diagonal matrix  $\mathbf{D} = [\mathbf{d}_{i,j}]$  where



$$\mathbf{d}_{i,j} = \begin{cases} \mathbf{a}_{i,j} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} .$$

### 3.2.1 Lanczos Method

8. Lanczos strategy : Do nothing to  $\mathbf{g}_{m+1}$ .

Davidson and Jacobi-Davidson methods are based on Lanczos method, which is a classical method and was published in 1950. The small matrix  $\mathbf{A}_m$  derived from Lanczos method has very convenient property - it is a tridiagonal matrix. This property is not shared by Davidson and Jacobi-Davidson methods. So, it is easier to calculate  $\mathbf{A}_m$  by using Lanczos method. Nevertheless, the convergence of Lanczos method is much slower than the other methods in general, and so we don't tell its detail.

### 3.2.2 Davidson Method

8. Davidson strategy :

$$\mathbf{g}_{m+1} = (\mathbf{D} - \lambda_m \mathbf{I})^{-1} \mathbf{r}_m. \quad (3.5)$$

Davidson method changes the direction of the residual vector by dividing each component of  $\mathbf{g}_{m+1}$  by  $\mathbf{a}_{i,i} - \lambda_m$ . This operation is sometimes called the diagonal preconditioning. The proof of convergence is given in [5].

### 3.2.3 Jacobi-Davidson Method

8. Jacobi-Davidson strategy :

$$\mathbf{g}_{m+1} = \alpha(\mathbf{D} - \lambda_m \mathbf{I})^{-1} \mathbf{x}_m - (\mathbf{D} - \lambda_m \mathbf{I})^{-1} \mathbf{r}_m, \quad (3.6)$$

where

$$\alpha = \frac{\mathbf{x}_m^T (\mathbf{D} - \lambda_m \mathbf{I})^{-1} \mathbf{r}_m}{\mathbf{x}_m^T (\mathbf{D} - \lambda_m \mathbf{I})^{-1} \mathbf{x}_m}. \quad (3.7)$$

The preconditioning in Jacobi-Davidson method is meant to find an approximate complement of the real eigenvector orthogonal to the Ritz vector [6], hence its has more direct geometrical interpretation than Davidson's.

## 3.3 Other Notations

### 3.3.1 Constructing Initial Vector $\mathbf{g}_1$

In general, a good initial choice of vector for  $\mathbf{g}_1$  can lead to faster convergence. Although there is no absolute way to generate it, we can provide a general way in Quantum Chemistry.

First of all, given a positive integer  $q$ , let the locational set

$$\mathcal{L} = \{\sigma : \mathbf{a}_{\sigma,\sigma} \text{ is one of the } q \text{ smallest elements in the diagonal part of } \mathbf{A}\}. \quad (3.8)$$

Let  $\mathcal{K}$  is spanned by  $\{\mathbf{e}_\sigma : \sigma \in \mathcal{L}\}$ . Define the projector  $\mathbf{P}$  onto  $\mathcal{K}$  as

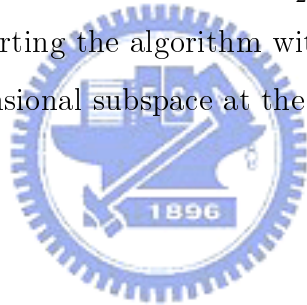
$$\mathbf{P} = \sum_{\sigma \in \mathcal{L}} \mathbf{e}_\sigma \mathbf{e}_\sigma^T. \quad (3.9)$$

There exists an eigenvector  $\mathbf{y}$  corresponding to the minimum eigenvalue  $\lambda$  of  $\mathbf{P}^T \mathbf{A} \mathbf{P}$  with restriction to  $\mathcal{K}$ . Then  $\mathbf{g}_1 = \mathbf{P} \mathbf{y}$  is a choice of the initial vector.

### 3.3.2 Restart Strategies

The CPU time of each step increases since the subspace is becoming larger and larger. The cost of each step comes mainly from the requirement of maintaining the orthogonality. If we do not wish to spend too much time for finding the extreme eigenvalue - restart strategies may solve this problem. For instance, after 20 steps, you can remove all the vectors  $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_{20}$  and take the Ritz vector  $\mathbf{x}_{20}$  as the initial vector  $\mathbf{g}_1$ .

Unfortunately, it may lead to reducing the speed of convergence or even a stagnation. A better way to solve this issue is to take more than one Ritz vectors as the starting space [6]. In the example above, you have the eigenvector  $\mathbf{y}_{20}$  corresponding to the minimum eigenvalue  $\lambda_{20}$  of  $\mathbf{P}_{20}^T \mathbf{A} \mathbf{P}_{20}$ . Also, there are eigenvectors  $\mathbf{y}'_{20}, \mathbf{y}''_{20}$  and  $\mathbf{y}'''_{20}$  corresponding to the 2nd, 3rd and 4th minimum eigenvalues  $\lambda'_{20}, \lambda''_{20}$  and  $\lambda'''_{20}$ , respectively. Then you have four Ritz vectors  $\mathbf{x}_{20}, \mathbf{x}'_{20}, \mathbf{x}''_{20}$  and  $\mathbf{x}'''_{20}$  obtained analogously to  $\mathbf{x}_{20} = \mathbf{P}_{20} \mathbf{y}_{20}$ . After restarting the algorithm with these four Ritz vectors as  $\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3$  and  $\mathbf{g}_4$ , you have 4 dimensional subspace at the beginning.



# Chapter 4

## Loss of Orthogonality

In Step 9 in Algorithm 3.1 we orthogonalize the vectors  $\mathbf{g}_1, \mathbf{g}_2, \dots, \mathbf{g}_m$  to make sure they are orthogonal. Unfortunately, if  $\mathbf{g}_{m+1}$  lies almost entirely in  $\mathcal{K}_m$ , then the vectors will be just approximately orthogonal because of finite precision of their numerical representation. We explain this by an easy example again below.

### 4.1 An Easy Example

Let


$$\mathbf{g}_1 = \begin{bmatrix} 0.8 \\ 0.6 \\ 0.0 \end{bmatrix} \quad \text{and} \quad \mathbf{g}_2 = \begin{bmatrix} -0.6 \\ 0.8 \\ 0.0 \end{bmatrix}.$$

Clearly,  $\mathbf{g}_1$  is orthogonal to  $\mathbf{g}_2$ . Suppose that

$$\mathbf{g}_3 = \begin{bmatrix} 1.0 \\ 1.0 \\ \alpha \end{bmatrix}$$

is the vector obtained from Step 8 in Algorithm 3.1, where  $\alpha$  is a small number. In Step 9,  $\mathbf{g}_3$  has to be orthogonalized with respect to  $\mathbf{g}_1$  and  $\mathbf{g}_2$ . All operations are performed in double precision finite arithmetics and the final vector  $\mathbf{g}_3$  is only approximately orthogonal to the previous vectors. We give the final vector  $\mathbf{g}_3$  for some  $\alpha$  below.

$$\begin{array}{cccc}
\alpha = 10^{-4} & \alpha = 10^{-8} & \alpha = 10^{-12} & \alpha = 10^{-16} \\
\Downarrow & \Downarrow & \Downarrow & \Downarrow \\
\begin{bmatrix} 4 \cdot 10^{-12} \\ 2 \cdot 10^{-12} \\ 1.0 \end{bmatrix} & \begin{bmatrix} 4 \cdot 10^{-8} \\ 2 \cdot 10^{-8} \\ 1.0 \end{bmatrix} & \begin{bmatrix} 4 \cdot 10^{-4} \\ 2 \cdot 10^{-4} \\ 1.0 \end{bmatrix} & \begin{bmatrix} 0.3619 \\ 0.2212 \\ 0.9056 \end{bmatrix}
\end{array} \quad (4.1)$$

The loss of orthogonality is more obvious while  $\alpha$  is smaller. It will inevitably lead to a wrong approximation of eigenvalues.

## 4.2 Examples using Davidson Method

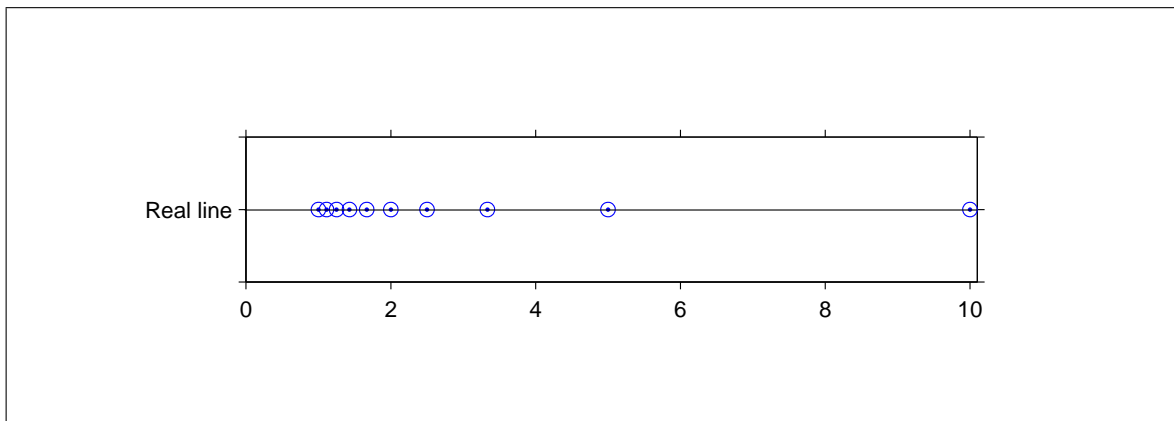
In this section, we show that for some matrices Davidson method fails to converge.  $\bar{\mathbf{D}}$ ,  $\hat{\mathbf{D}}$ , and  $\tilde{\mathbf{D}}$  mentioned below are diagonal matrices, and we always use the same starting vector  $\mathbf{t}_1 = (0, 0, \dots, 0, 0.8, 0.6)^T$ . In Example 1 and Example 2, we show that the problems can occur when the extreme eigenvalue lies within the discrete and clustered part of spectrum of  $\mathbf{A}$  respectively. In Example 3, we show that in some cases the converged the extreme eigenvalue is completely wrong even if the residual value is small enough.

*Example 1.*

Let  $\bar{\mathbf{A}}$  be of order 200 defined by

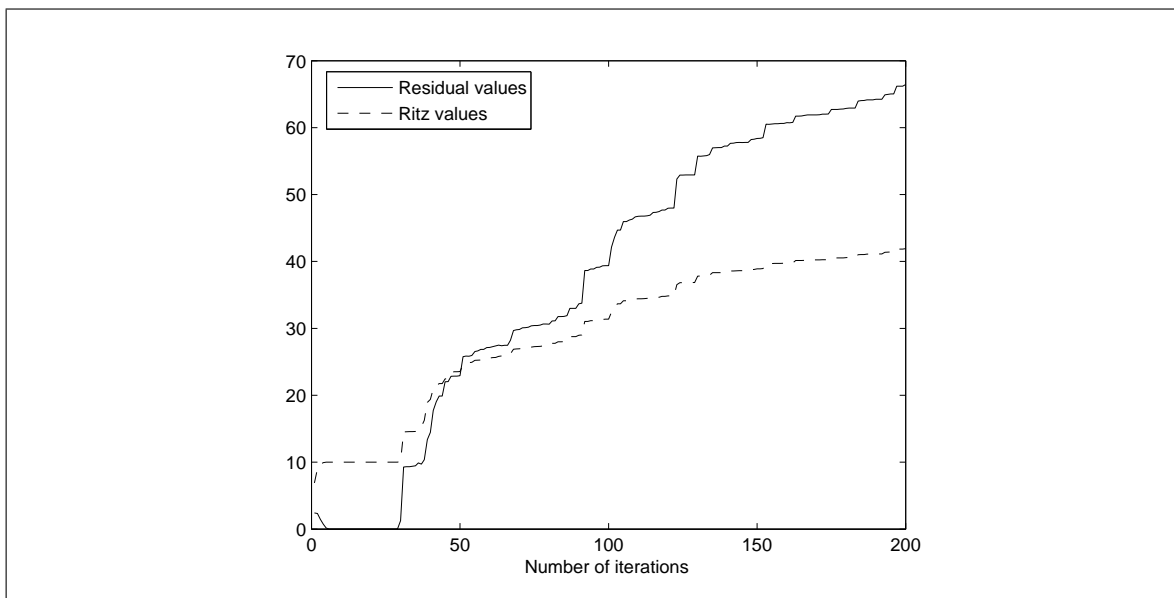
$$\bar{\mathbf{a}}_{i,j} = \begin{cases} \frac{10}{201-i} & \text{if } i = j \\ 0.1 & \text{if } |i - j| = 1 \\ 0 & \text{if } |i - j| > 1 \end{cases} . \quad (4.2)$$

The diagonal part  $\{\bar{\mathbf{a}}_{i,i}\}_{i=1}^{200}$  of  $\bar{\mathbf{A}}$  increases monotonically. By Gerschgorin's Theorem 1 all the radii are only 0.1, hence the distribution of spectrum is similar to the one of diagonal part of  $\bar{\mathbf{A}}$ . So the maximum eigenvalue is in the discrete part of spectrum (Figure 4.1). The overlap matrix  $\mathbf{O}_m$  at step  $m$  is defined by  $\langle \mathbf{g}_i, \mathbf{g}_j \rangle$ , where  $\langle \cdot, \cdot \rangle$  is the usual inner product. Normally, the overlap matrix in each step should be very close to the identity matrix normally. In Figure 4.2, we can see that the residual values and



**Figure 4.1** Location of the largest 10 eigenvalues.

Ritz values blow up while finding the maximum eigenvalue of  $\bar{\mathbf{A}}$ . Since overlap matrices



**Figure 4.2** Residual values and Ritz values from Example 1.

are symmetric, we just show the upper triangular part of  $\mathbf{O}_{30}$  excluding the diagonal elements in Figure 4.3. As you can see, it is very different from the identity matrix.

*Example 2.*

Since  $\bar{\mathbf{A}}$  is symmetric, we can find an eigendecomposition:  $\bar{\mathbf{A}} = \mathbf{V}\bar{\mathbf{D}}\mathbf{V}^T$ . Let  $\hat{\mathbf{A}} = \mathbf{V}\hat{\mathbf{D}}\mathbf{V}^T$  where  $\hat{\mathbf{D}}$  is given by  $\hat{\mathbf{d}}_{i,i} = \frac{10}{i} - 1$  for  $i = 1$  to 200. The portrait of  $\hat{\mathbf{A}}$  is plotted in Figure 4.4; it is diagonally dominant. Constructing  $\hat{\mathbf{A}}$  in this way implies

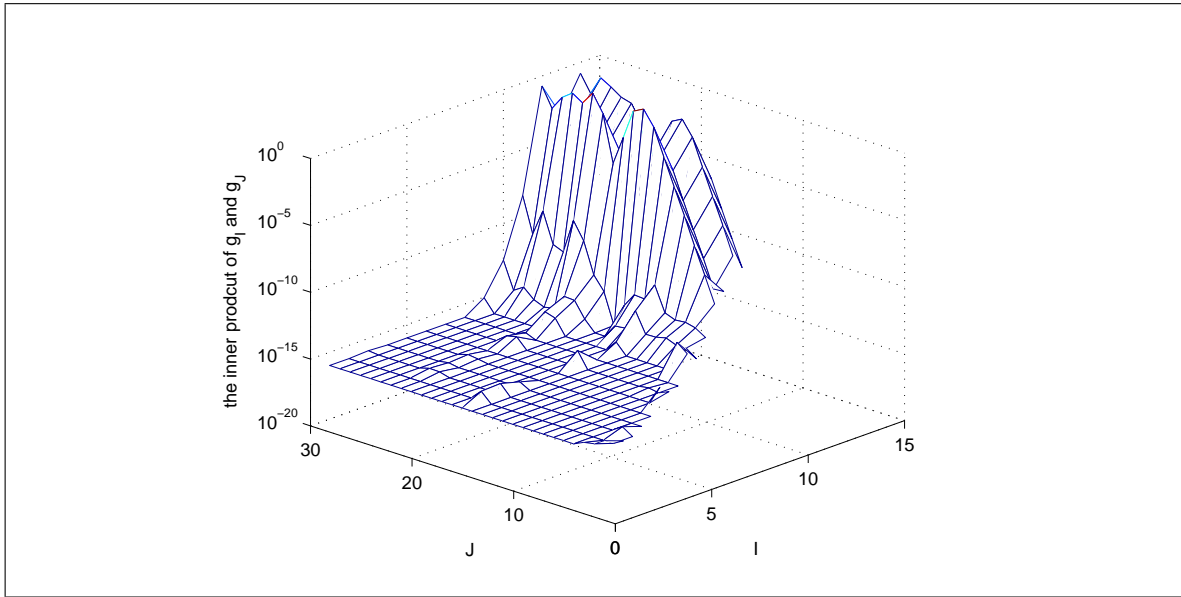


Figure 4.3 Overlap matrix  $\mathbf{O}_{30}$  from Example 1.

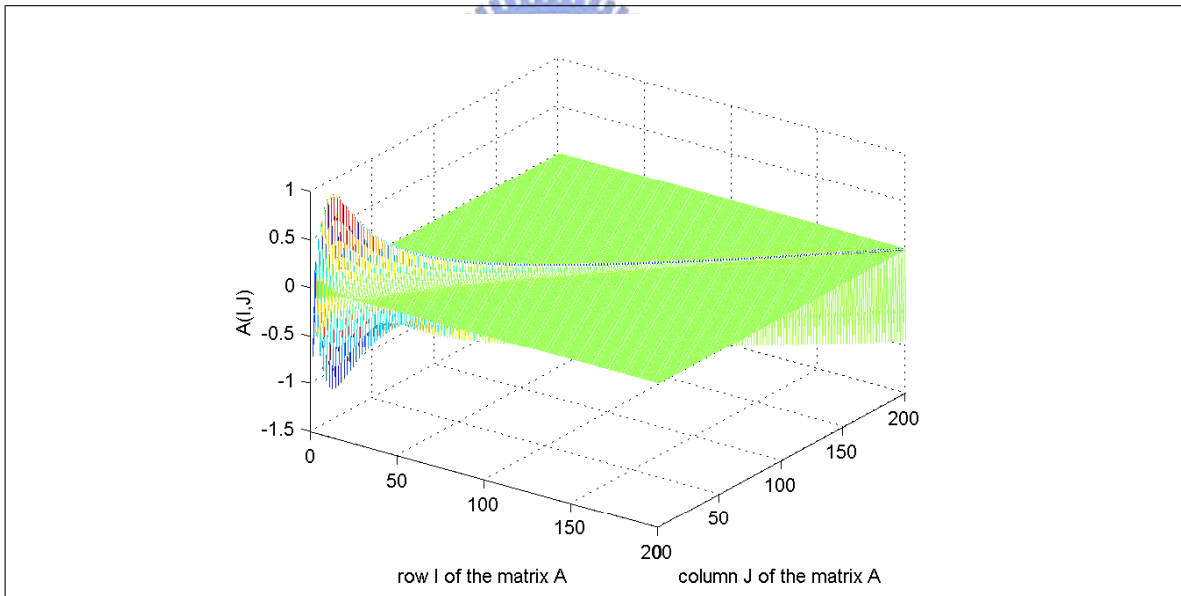


Figure 4.4 Graphical presentation of  $\hat{\mathbf{A}}$ .

that the minimum eigenvalue of  $\hat{\mathbf{A}}$  is  $-0.95$ . By applying Gerschgorin's theorems again, we consider a matrix whose minimum eigenvalue  $-0.95$  contained in the clustered part of spectrum (Figure 4.5). A similar situation like Example 1, this case also diverges when looking for the minimum eigenvalue (Figure 4.6).

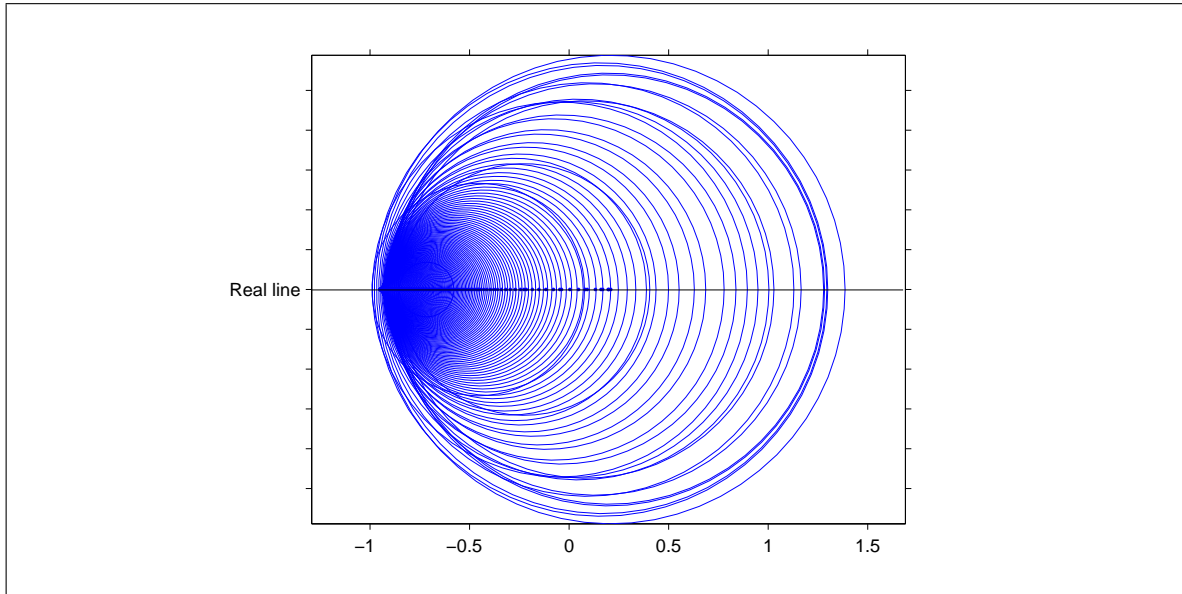


Figure 4.5 Location of all eigenvalues of  $\tilde{\mathbf{A}}$ .

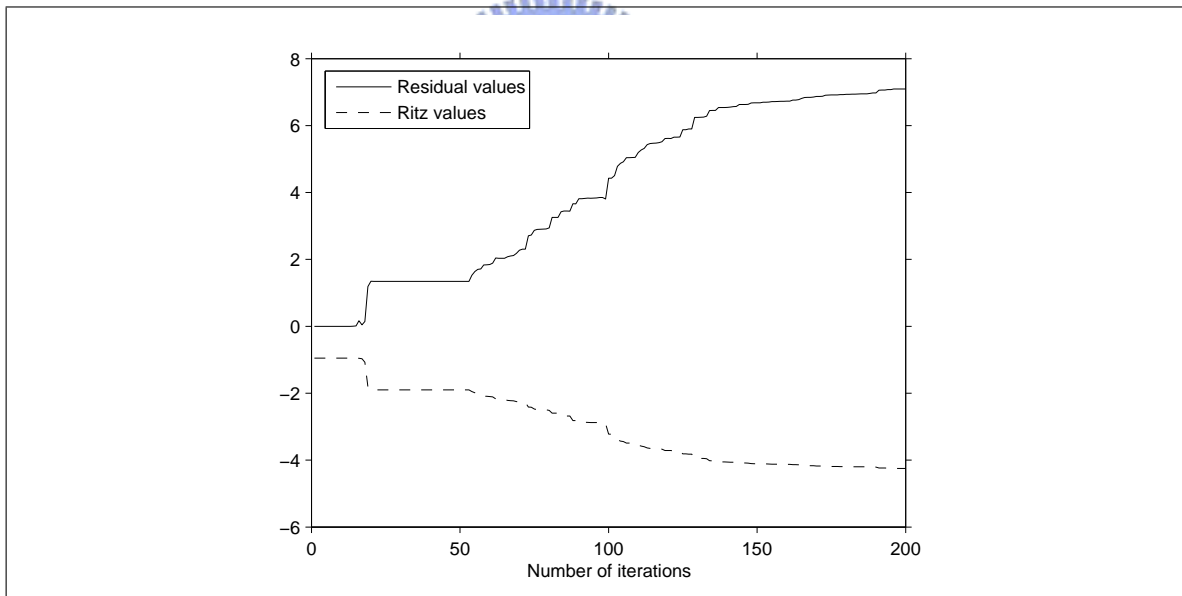
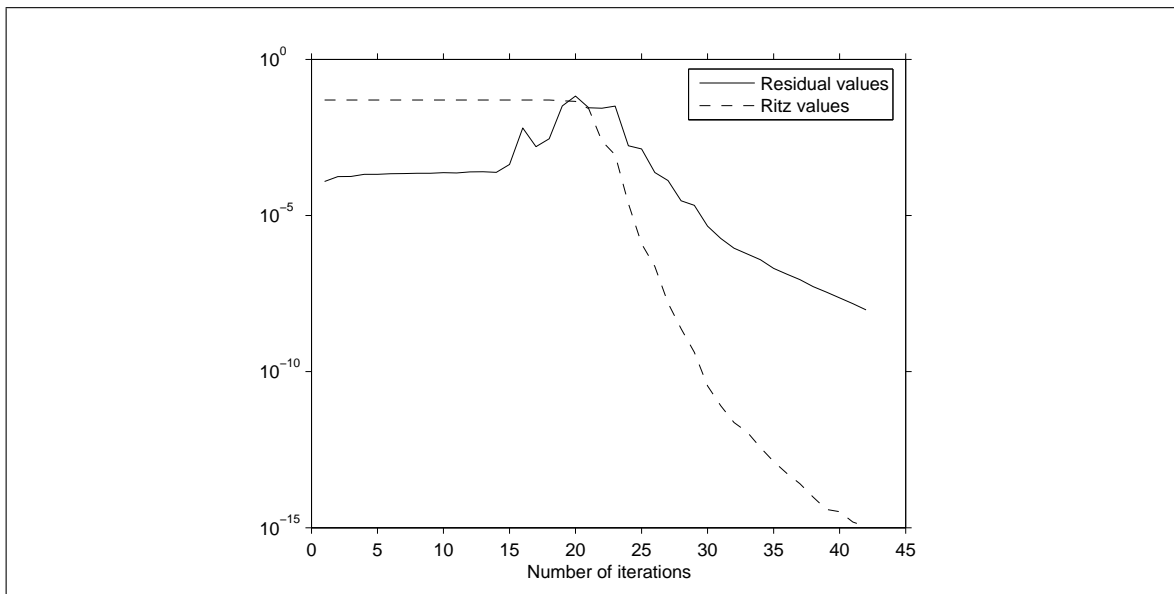


Figure 4.6 Residual values and Ritz values from Example 2.

### Example 3.

An interesting thing happens when we try to find the minimum eigenvalue of  $\tilde{\mathbf{A}} = \mathbf{V}\tilde{\mathbf{D}}\mathbf{V}^T$ .  $\mathbf{V}$  is the one in Example 2 and  $\tilde{\mathbf{D}}$  is given by  $\tilde{\mathbf{d}}_{i,i} = \frac{10}{i}$  for  $i = 1$  to 200. The portrait of  $\tilde{\mathbf{A}}$  is similar to the one of  $\hat{\mathbf{A}}$  and so it is dominant diagonally, too. From





**Figure 4.7** Residual values and Ritz values from Example 3.

Figure 4.7, it looks as if Davidson method has converged. Unfortunately, it is a "fake convergence" because the minimum eigenvalue should be 0.05. We use the notation from Chapter 3 to explain this situation. At step  $m$ , we have an approximate eigenpair  $(\lambda_m, \mathbf{y}_m)$ .  $\lambda_m$  can be represented as

$$\lambda_m = \mathbf{y}_m^T \lambda_m \mathbf{y}_m = \mathbf{y}_m^T \mathbf{P}_m^T \tilde{\mathbf{A}} \mathbf{P}_m \mathbf{y}_m = (\mathbf{P}_m \mathbf{y}_m)^T \tilde{\mathbf{A}} \mathbf{P}_m \mathbf{y}_m.$$

So,  $\lambda_m > 0$  since  $\tilde{\mathbf{A}}$  is positive definite. (The eigenvalues of  $\tilde{\mathbf{A}}$  are all positive.) That's why  $\hat{\mathbf{A}}$  and  $\tilde{\mathbf{A}}$  are similar, but the Ritz values here don't decrease like the ones in Figure 4.7.

If the orthogonalization in Step 9 of Algorithm 3.1 is done twice, Davidson method will converge to the proper extreme eigenvalues for all these three matrices deduct here.

# Chapter 5

## The Sweep Method

Davidson method usually converges within several iterations if the matrix is sufficiently diagonally dominant. Nevertheless, sometimes the convergence is slower. When the Ritz value is very close to some eigenvalue, then it may require one hundred steps to converge. In this chapter, we will discuss a modified method named as the sweep method to reduce the number of steps.

### 5.1 Jacobi-sweep Method

Before introducing the sweep method, we discuss its main idea first - Jacobi-sweep method. Jacobi-sweep method is based on Jacobi rotations [7, pages 426–438]. It is used to search for the extreme eigenvalues.

In the space  $\mathbf{R}^n$ , there exists an orthonormal basis  $\mathcal{B} = \{\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_n\}$ . Dividing  $\mathcal{B}$  into subsets of  $d$  elements yields a partition of  $\mathcal{B} : \mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_{\lceil \frac{n}{d} \rceil}$ . (the last subset  $\mathcal{B}_{\lceil \frac{n}{d} \rceil}$  may contain less vectors than  $d$ .) When Jacobi-sweep method is executed, we need a normalized starting vector  $\mathbf{s}$  with its first  $d$  components equal to zero. We can brief characterize this method in the following way.

For  $i = 1$  to  $\lceil \frac{n}{d} \rceil$ , do

1. Define the projector  $\mathbf{P}$  onto the subspace spanned by  $\mathbf{s}$  and  $\mathcal{B}_i$  as

$$\mathbf{P} = \mathbf{s}\mathbf{s}^T + \sum_{\mathbf{e}_j \in \mathcal{B}_i} \mathbf{e}_j \mathbf{e}_j^T. \quad (5.1)$$

2. Calculate the eigenvector  $\mathbf{y}$  corresponding to the minimum eigenvalue  $\lambda$  of  $\mathbf{P}^T \mathbf{A} \mathbf{P}$  with restriction to the space spanned by the vectors in  $\mathcal{B}_i$ , and redefine  $\mathbf{s} = \mathbf{P}\mathbf{y}$ .

3. If

$$\mathbf{A}\mathbf{s} \doteq \lambda\mathbf{s}, \quad (5.2)$$

then  $(\lambda, \mathbf{s})$  will be the approximate minimum eigenpair of  $\mathbf{A}$ . Otherwise, make next  $d$  components of  $\mathbf{s}$  equal to zero, and then normalize it.

end do

We call this procedure "one sweep" or "sweeping once". After this process is executed one time, there may be no convergence. Then, we define the final vector  $\mathbf{s}$  as the starting vector of the next sweep. It may converge to the sought eigenvalue by a large number of sweeps, and therefore it may prove to be useless for practical reasons. In the next section, we discuss about how to use Jacobi-sweep method in practice.

## 5.2 Main Idea

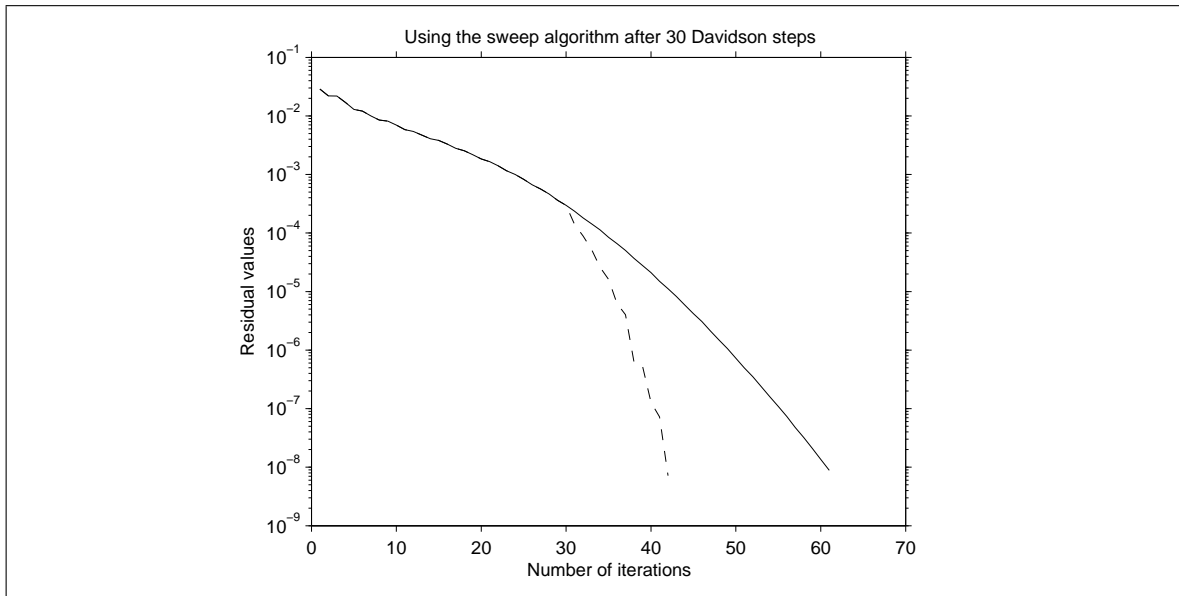
The main structure of the sweep method is similar to the methods mentioned in Chapter 3. The main difference is Step 8 in Algorithm 3.1. The strategy in the sweep method is as follows:

8. The sweep strategy : Sweep the Ritz vector a fixed number of times and define it as  $\mathbf{g}_{m+1}$ .

In Davidson and Jacobi-Davidson methods, the residual vector is modified in Step 9, however here we change the direction of the Ritz vector by relaxing it by the sweep method. That is a quite different approach to construct the next Ritz vector. We show some results obtained using this approach in the next section.

## 5.3 Results

We found that the sweep method has faster convergence than Davidson and Jacobi-Davidson methods. This phenomenon is demonstrated using the matrix  $\bar{\mathbf{A}}$  from Example 1 in Chapter 4, but here we search for the minimum eigenvalue.



**Figure 5.1** Convergence for the sweep method is faster than for Davidson method.

In Figure 5.1, the dashed line represents the convergence history obtained with combined Davidson and the sweep methods. It uses Davidson method in the first 30 steps, and the sweep method later. The solid line represents the convergence history obtained when only Davidson method is used. We see the dashed line goes down faster than the solid one, which means that the sweep method has faster convergence in this case.

In Figure 5.2, we use the sweep and Davidson methods for the matrix  $\bar{\mathbf{A}}$ . You can see the dashed line is below the solid one, which means that the sweep method needs fewer steps to converge. In the present implementation, the cost of the preconditioner for the sweep method is more expensive than for Davidson and Jacobi-Davidson methods. Therefore these methods compete for time. It is possible that the sweep method is faster than the other methods since it needs much fewer iterations, even though it costs more time for preconditioning.

Now we introduce one kind of matrix that is called the band matrix. For a band matrix  $\mathbf{B}$ , it has nonzero elements  $\mathbf{b}_{i,j}$  for  $0 \leq |i - j| \leq \omega$ , where  $\omega$  is called the

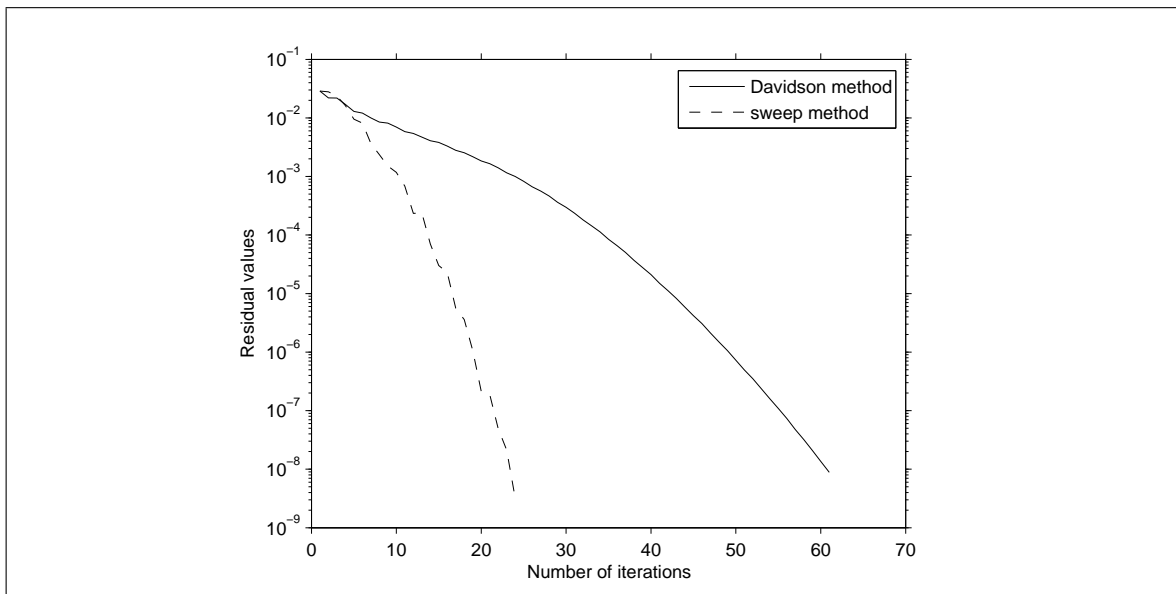


Figure 5.2  $\bar{\mathbf{A}}$  using the sweep and Davidson methods.

bandwidth of  $\mathbf{B}$ . Our  $\mathbf{B}$  is also diagonally dominant. Suppose

$$\mathcal{R} = \{(i, j) : \mathbf{b}_{i,j} \text{ is in the band region and } i \neq j\}. \quad (5.3)$$

To define  $\mathbf{b}_{h,k}$  for  $(h, k) \in \mathcal{R}$ , we use an array with  $n$  components called *offdiag*. Each component of *offdiag* is generated as a uniform random number between -0.5 and 0.5. If  $|\text{offdiag}(h) \times \text{offdiag}(k)|$  is larger than a specified value  $\delta$ , then  $\mathbf{b}_{h,k} = 0$ . Otherwise,

$$\mathbf{b}_{h,k} = \zeta \times \text{offdiag}(h) \times \text{offdiag}(k), \quad (5.4)$$

where  $\zeta$  is a positive real number. When  $\delta$  is larger, the number of nonzero elements in the band region except the diagonal is larger. By controlling  $\delta$ , we can change the percentage  $\rho$  of the nonzero elements in the band region except the diagonal. Suppose

$$\mathcal{Z} = \{(i, j) \in \mathcal{R} : |\text{offdiag}(i) \times \text{offdiag}(j)| \leq \delta\}, \quad (5.5)$$

then let  $\alpha = \frac{\sum_{(i,j) \in \mathcal{Z}} |\mathbf{b}_{i,j}|}{|\mathcal{Z}|}$  represent the average magnitude of the nonzero elements in the band region except the diagonal. Since

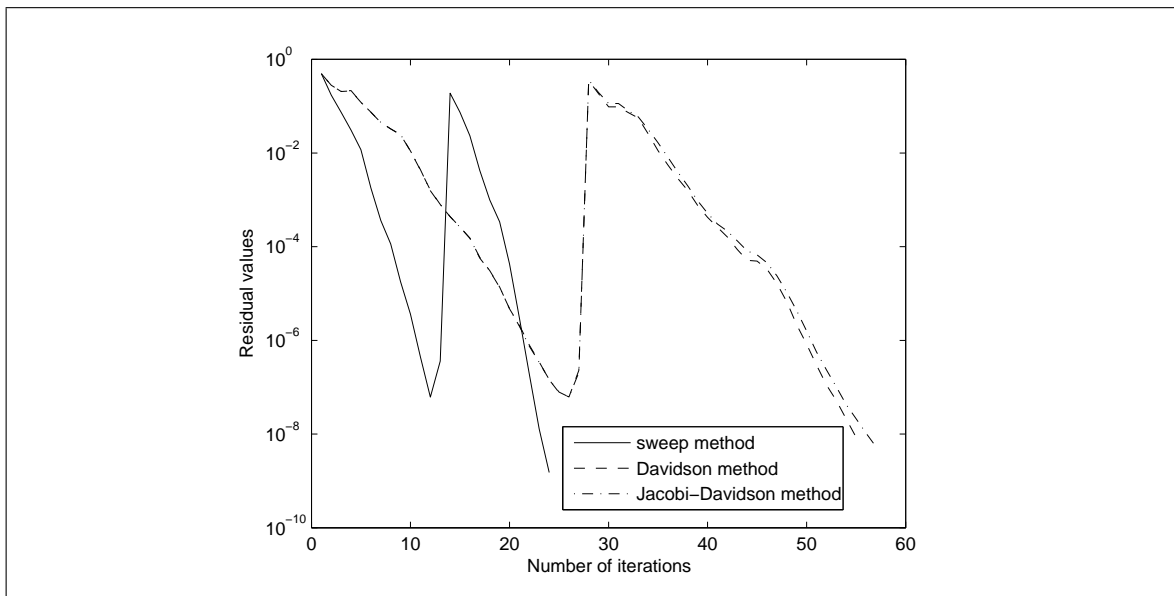
$$\alpha = \frac{\sum_{(i,j) \in \mathcal{Z}} |\mathbf{b}_{i,j}|}{|\mathcal{Z}|} = \zeta \frac{\sum_{(i,j) \in \mathcal{Z}} |\text{offdiag}(i) \times \text{offdiag}(j)|}{|\mathcal{Z}|}, \quad (5.6)$$

$\alpha$  can be set to some value simply by changing  $\zeta$ . Our band matrix  $\mathbf{B}$  is defined as below:

$$\mathbf{b}_{i,j} = \begin{cases} \beta & \text{if } i = j \\ \gamma & \text{for the probability } \rho \\ 0 & \text{for the probability } 100\% - \rho \\ 0 & \text{if } |i - j| > \omega \end{cases},$$

where  $\beta$  is a uniform random number between -10 and 10, and  $\gamma$  is  $\zeta \times \text{offdiag}(i) \times \text{offdiag}(j)$  where  $\zeta$  is some positive value designated by  $\alpha$ . We discuss the band matrix  $\mathbf{B}$  of small order  $10^4$  with  $\omega = 500$  and large order  $10^6$  with  $\omega = 1000$  respectively.

For small order, there are 1000 generated matrices of the form  $\mathbf{B}$  for each  $\rho = 10\%, 20\%, \dots, 50\%$  and each  $\alpha = 0.01, 0.02, \dots, 0.05$ . We use the sweep, Davidson, and Jacobi-Davidson methods to find the minimum eigenvalue of these matrices and consider which method is faster. For the sweep method, we sweep Ritz vectors twice for the sweep method. Unfortunately, the sweep method has a serious drawback - the loss of orthogonality. It needs two reorthogonalizations usually. The reason for this is because the Ritz vector may change only a little after preconditioning, so the preconditioned vector is very closed to the original subspace  $\mathcal{K}_m$ . Because of this trouble, we use one reorthogonalization for Davidson and Jacobi-Davidson methods but two for the sweep method. We also restart the procedure every 30 iterations to reduce the computational time (Section 3.3.2). In Appendix B, we give results obtained with these methods. From Table B.1 to Table B.5, the average execution time for the sweep method is smaller than for the other two methods. From Table B.6 to Table B.8, the winning times denotes the number of being the fastest from 1000 matrices. As  $\alpha$  and  $\rho$  increase, the winning times of these two methods decrease on average, i.e., the sweep method is the fastest in most cases. The average steps of the sweep method is less than half the ones of the other two methods. We take a sample for  $\rho = 30\%$  and  $\alpha = 0.03$ , and the result is the following. In this case, they have similar curves, but the one for the sweep method is sharper. Because of this phenomenon, the time for the sweep method is shorter, even it needs more time for preconditioning.



**Figure 5.3** A sample for  $\rho = 30\%$  and  $\alpha = 0.03$ .

For large order, there are 100 generated matrices for each  $\rho = 10\%, 12.5\%, \dots, 20\%$  and each  $\alpha = 0.01, 0.0125, \dots, 0.02$ . The settings are almost the same with small matrices except restarting every 10 iterations and sweeping Ritz vectors 3 times. Similar results are presented in Appendix C, and the sweep method is faster again.

## 5.4 Conclusions

From the presented results, it is clear that the sweep method is faster than Davidson and Jacobi-Davidson methods for most of the studied small and large matrices. The superiority of the sweep method is more obvious as  $\rho$  and  $\alpha$  increase, i.e., as the numerical cost increases. That means the sweep method may be even better for higher  $\rho$  and  $\alpha$ .

From Table 5.1, there are four common cases with the same  $\rho$  and  $\alpha$  for small and large matrices. We just show the ratio of the average time for the sweep method to the average time for Davidson method since the results of Davidson and Jacobi-Davidson are similar. From small to large matrices, the ratio increases and is larger than 1, so it

is possible that the ratio for matrices of higher order is even larger. We can guess that the sweep method is faster than Davidson and Jacobi-Davidson methods for matrices of higher order.

**Table 5.1** The ratio between the average time for Davidson method and the average time for the sweep method for various values of  $\rho$  and  $\alpha$ .

The probability ( $\rho$ )	The average ( $\alpha$ )	Small matrices	Large matrices
10%	0.01	1.094086	1.734938
10%	0.02	1.34382	1.76378
20%	0.01	1.111489	1.66922
20%	0.02	1.422389	1.990131

The sweep method indicates two things. First, the iterative methods mentioned in the thesis always change the direction of residual vectors, however the sweep method changes directly the Ritz vector. Although it needs to deal with the loss of orthogonality, the sweep method still costs less time to converge.

Second, it costs just little time to precondition a vector for general iterative methods. The sweep method tells us that we may need less steps to converge the extreme eigenvalues for preconditioning some specified vector more time. So it is competitive for the sweep method since it reduces the number of iterations in general. The sweep method gives us a quite different way to search the eigenvalue.



# Bibliography

- [1] B. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, 1st edition, 1980.
- [2] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS, 1st edition, 1996.
- [3] J.H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon, 1st edition, 1965.
- [4] A.R. Gourlay and G.A. Watson. *Computational Methods for Matrix Eigenproblems*. John Wiley and Sons, 1st edition, 1973.
- [5] M. Crouzeix, B. Philippe, and M. Sadkane. The Davidson Method. *SIAM J. SCI COMPUT*, 15:63–65, 1994.
- [6] G.L.G. Sleijpen and H.A. Van Der Vorst. The Jacobi-Davidson Method for Eigenvalue Problems and Its Relation with Accelerated Inexact Newton Schemes. *Comput. Appl. Math*, 3:377–389, 1996.
- [7] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins, 3rd edition, 1996.



# Appendix A

## Program Codes

### A.1 Main Program

```
! ORDER FOR ORDER OF A
! P FOR THE COLUMNS OF SUBA
! LIMIT FOR ROWS OF G
! TYPE=1:MAXIMUM AND TYPE=0:MINIMUM
! E_VECTPR IS THE INITIAL VECTOR FOR INPUT AND EIGENVECTOR FOR OUTPUT
! E_VALUE IS THE MIN EIGENVALUE IF TYPE=0 AND MAX ONE IF TYPE=1
! =====
subroutine general(way,diag,offdiag,order,can,limit,type,repeat,e_vector,
e_value,show,step,sweeps,population,prob1,prob2,offarray,nonzero,total,
position,blockA,block_offarray,block_nonzero,block_total,block_position)

implicit none

character(1):: way
integer:: order,limit,type,can,sweeps,step,check,repeat,show,population
integer:: i, j, k, m, q, r, INFO, LWORK, total, block_total
real(kind=8):: ddot,dnrm2,sum,maks,e_value,prob1,prob2
real(kind=8), dimension(:), allocatable:: WORK
real(kind=8), dimension(order):: Avector, diag, e_vector, WR, offdiag, res
real(kind=8), dimension(order,can):: blockA
```

```

real(kind=8), dimension(limit,limit):: F, GAG
real(kind=8), dimension(limit,order):: G, GA
integer, dimension(order+1):: position, block_position
integer, dimension(total):: offarray
integer, dimension(block_total):: block_offarray
real(kind=8), dimension(total):: nonzero
real(kind=8), dimension(block_total):: block_nonzero

step=1
check=1

! PRODUCE NORMALIZED INITIAL VECTOR
! =====
sum=dnorm2(order,e_vector,1)
call dscal(order,1.0d0/sum,e_vector,1)
call dcopy(order,e_vector,1,G,limit)

do while(step<=order)

    ! DETERMINE AND DIAGONALIZE GAG
    ! =====
    do i=1,order
        GA(check,i)=0.0d0
        do j=position(i),position(i+1)-1
            GA(check,i)=GA(check,i)+nonzero(j)*G(check,offarray(j))
        end do
        GA(check,i)=GA(check,i)+diag(i)*G(check,i)
    end do
end do

```

```

call dgemv('N',check,order,1.0d0,GA,limit,G(check,1),limit,0.0d0,
          GAG(1,check),1)

do i=1,check
    call dcopy(i,GAG(1,i),1,F(1,i),1)
end do

LWORK=-1
ALLOCATE(WORK(1))
call dsyev('V','U',check,F,limit,WR,WORK,LWORK,INFO)
LWORK=INT(WORK(1))
DEALLOCATE(WORK)
ALLOCATE(WORK(LWORK))
call dsyev('V','U',check,F,limit,WR,WORK,LWORK,INFO)
DEALLOCATE(WORK)

! PRODUCE RITZ VECTOR
! =====
r=(1-type)+check*type
e_value=WR(r)

call dgemv('T',check,order,1.0d0,G,limit,F(1,r),1,0.0d0,e_vector,1)
call dgemv('T',check,order,1.0d0,GA,limit,F(1,r),1,0.0d0,Avector,1)

do i=1,order
    res(i)=Avector(i)-e_value*e_vector(i)
end do
sum=dnrm2(order,res,1)
if(show==1) write(*,'(i5,4ES25.16)')step,sum,e_value

```

```
! ORTHOGONALIZATION
! =====
if(sum<1.0d-8) then
    if(show==0)write(*,'(i5,2ES25.16,25X,ES25.16)')step,sum,e_value
    exit

else if(check<limit) then
    step=step+1
    check=check+1

select case(way)
case('D')
    call davidson(order,res,Avector,diag,e_value,sum)
    call dscal(order,1.0d0/sum,Avector,1)
case('J')
    call jacobi_davidson(order,res,e_vector,Avector,diag,
        e_value,sum)
    call dscal(order,1.0d0/sum,Avector,1)
case('S')
    call jsweep(diag,offdiag,order,type,can,e_value,e_vector,
        Avector,sweeps,offarray,nonzero,total,position,
        population,prob1,prob2,blockA,block_offarray,
        block_nnzero,block_total,block_position)
    call dcopy(order,e_vector,1,Avector,1)
end select

do q=1,repeat
    call dcopy(check-1,0.0d0,0,WR,1)
```

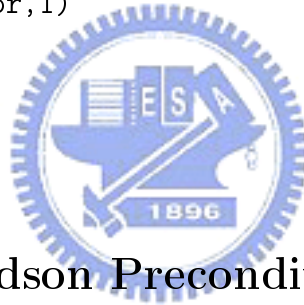
```
do j=1,check-1
  maks=0.0d0
  do i=1,check-1
    sum=0.0d0
    if(WR(i)==0.0d0) then
      sum=ddot(order,G(i,1),limit,Avector,1)
      if(abs(sum)>=abs(maks)) then
        maks=sum
        m=i
      end if
    end if
  end do
  WR(m)=1.0d0
  call daxpy(order,-maks,G(m,1),limit,Avector,1)
  sum=dnrm2(order,Avector,1)
  call dscal(order,1.0d0/sum,Avector,1)
end do
end do
call dcopy(order,Avector,1,G(check,1),limit)
else
  step=step+1
  check=1
  call dcopy(order,e_vector,1,G,limit)
end if
end do
end subroutine
```

## A.2 Davidson Preconditioner

```
implicit none
integer:: i, order
real(kind=8):: e_value, sum, dnorm2
real(kind=8), dimension(order):: res, Avector, diag

do i=1,order
  Avector(i)=res(i)/(diag(i)-e_value)
end do

sum=dnorm2(order,Avector,1)
end subroutine
```



## A.3 Jacobi-Davidson Preconditioner

```
subroutine jacobi_davidson(order,res,e_vector,Avector,diag,e_value,sum)
implicit none
integer:: i, order
real(kind=8):: e_value, sum, dnorm2, ddot
real(kind=8), dimension(order):: e_vector, Avector, diag, D, res

do i=1,order
  D(i)=e_vector(i)/(diag(i)-e_value)
  Avector(i)=res(i)/(diag(i)-e_value)
end do

sum=ddot(order,D,1,e_vector,1)
```

```

sum=ddot(order,Avector,1,e_vector,1)/sum
do i=1,order
    Avector(i)=sum*D(i)-Avector(i)
end do
sum=dnrm2(order,Avector,1)
end subroutine

```

## A.4 Sweep Preconditioner

```

subroutine
jsweep(diag,offdiag,order,type,can,e_value,e_vector,Avector,sweeps,offarray,
nonzero,total,position,population,prob1,prob2,blockA,block_offarray,
block_nonzero,block_total,block_position)

! ORDER : THE ORDER OF MATRIX
! TYPE : TYPE=0, THE MIN EIGENVALUE ; TYPE=1, THE MAX EIGENVALUE
! CAN : THE NUMBER TO DIAGONALIZE ONCE
! SWEEPS : LIMIT THE NUMBER OF SWEEPS
! =====

implicit none

integer:: i, j, start_can, p, var, LWORK, INFO, count, sweeps, row, column,
        population, total, block_total, order, type, can
real(kind=8):: norm,e_value,dnrm2,ddot,coef,sum,prob1,prob2,a,b,c,d,e
real(kind=8), dimension(can):: adv
real(kind=8), dimension(order):: WR, res, e_vector, Avector, diag, offdiag
real(kind=8), dimension(can+1,can+1):: GAG
real(kind=8), dimension(order,can):: blockA

```



```

real(kind=8), dimension(:), allocatable :: WORK
integer, dimension(order+1):: position, block_position
integer, dimension(total):: offarray
real(kind=8), dimension(total):: nonzero
integer, dimension(block_total):: block_offarray
real(kind=8), dimension(block_total):: block_nonzero

var=can
start_can=1
count=0
coef=1.0d0

! THE MAIN PROGRAM
! =====
do while(start_can<=order)

! CONSTRUCT GAG(1,1)
! =====
100 call dcopy(var,0.0d0,0,adv,1)
do i=1,var
row=start_can-1+i
do j=block_position(row),block_position(row+1)-1
adv(i)=adv(i)+block_nonzero(j)*e_vector(block_offarray(j))
end do
end do

if(Avector(start_can)==adv(1).and.can==1) then
if(start_can<=order-1) then
start_can=start_can+1

```



```

    else
      start_can=1
      count=count+1
      sum=dnrm2(order,e_vector,1)
      call dscal(order,1.0d0/sum,e_vector,1)
      coef=1.0d0
    end if
    go to 100
  end if

  GAG(1,1)=ddot(var,Avector(start_can),1,e_vector(start_can),1)
  GAG(1,1)=e_value-2.0d0*GAG(1,1)/(coef*coef)
      +ddot(var,adv,1,e_vector(start_can),1)/(coef*coef)

! NORM OF THE STARTING VECTOR
! =====
  norm=0.0d0
  do i=0,var-1
    norm=norm+e_vector(start_can+i)*e_vector(start_can+i)/(coef*coef)
  end do
  norm=sqrt(1.0d0-norm)

! CONSTRUCT GAG
! =====
  GAG(1,1)=GAG(1,1)/(norm*norm)
  do i=0,var-1
    GAG(1,2+i)=(Avector(start_can+i)-adv(1+i))/(norm*coef)
  end do

```



```

do i=0,var-1
  do j=i+1,var
    GAG(2+i,j+1)=blockA(start_can+i,j)
  end do
end do

! DIAGONALIZE
! =====
if(can>=1) then
  LWORK=-1
  ALLOCATE(WORK(1))
  call dsyev('V','U',var+1,GAG,can+1,WR,WORK,LWORK,INFO)
  LWORK=INT(WORK(1))
  DEALLOCATE(WORK)
  ALLOCATE(WORK(LWORK))
  call dsyev('V','U',var+1,GAG,can+1,WR,WORK,LWORK,INFO)
  DEALLOCATE(WORK)
else
  if(type==0) then
    a=GAG(2,2)-GAG(1,1)
    b=GAG(1,2)*GAG(1,2)
    c=sqrt(a*a+4.0d0*b)
    WR(1)=(GAG(1,1)+GAG(2,2)-c)/2.0d0
    d=a+c
    e=sqrt(2.0d0*(a*d+4.0d0*b))
    GAG(1,1)=d/e
    GAG(2,1)=-2.0d0*GAG(1,2)/e
  else
    a=GAG(2,2)-GAG(1,1)

```

```

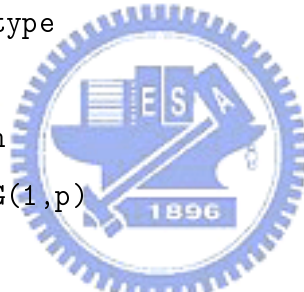
    b=GAG(1,2)*GAG(1,2)
    c=sqrt(a*a+4.0d0*b)
    WR(1)=(GAG(1,1)+GAG(2,2)+c)/2.0d0
    d=a-c
    e=sqrt(2.0d0*(a*d+4.0d0*b))
    GAG(1,1)=d/e
    GAG(2,1)=-2.0d0*GAG(1,2)/e
  end if
end if

! RITZ VECTOR
! =====
p=(1-type)+(var+1)*type
e_value=WR(p)
if(GAG(1,p)/=0) then
  coef=coef*norm/GAG(1,p)
else
  call dcopy(order,0.0d0,0,e_vector,1)
  coef=1.0d0
end if

do i=0,var-1
  e_vector(i+start_can)=GAG(i+2,p)*coef
end do

if(start_can>=order-can+1) then
  count=count+1
  sum=dnrm2(order,e_vector,1)
  call dscal(order,1.0d0/sum,e_vector,1)

```



```

    coef=1.0d0
end if

if(sweeps==count) then
    exit
else
    if(start_can<=order-2*can+1) then
        start_can=start_can+can
    else if(order-2*can+1<start_can.and.start_can<=order-can) then
        start_can=start_can+can
        var=order-start_can+1
    else
        start_can=1
        var=can
    end if

! CALCULATE AVECTOR(START_CAN)~AVECTOR(START_CAN-1+VAR)
! =====
    do i=1,var
        row=start_can-1+i
        Avector(row)=0.0d0
        do j=position(row),position(row+1)-1
            Avector(row)=Avector(row)+nonzero(j)*e_vector(offarray(j))
        end do
        Avector(row)=Avector(row)+diag(row)*e_vector(row)
    end do
end if
end do
end subroutine

```



# Appendix B

## Numerical Results for Small Matrices

**Table B.1** The average time for the probability  $\rho = 10\%$ .

<b>The average</b> ( $\alpha$ )	<b>Sweep method</b> (sec)	<b>Davidson</b> (sec)	<b>Jacobi-Davidson</b> (sec)
0.01	0.062836	0.068748	0.068764
0.02	0.09231	0.124048	0.125003
0.03	0.135992	0.216126	0.219582
0.04	0.185576	0.317008	0.318536
0.05	0.224782	0.39046	0.395705

**Table B.2** The average time for the probability  $\rho = 20\%$ .

<b>The average</b> ( $\alpha$ )	<b>Sweep method</b> (sec)	<b>Davidson</b> (sec)	<b>Jacobi-Davidson</b> (sec)
0.01	0.101974	0.113343	0.114015
0.02	0.197756	0.281286	0.28361
0.03	0.316548	0.499815	0.503755
0.04	0.390496	0.640088	0.64588
0.05	0.432939	0.733578	0.73815

**Table B.3** The average time for the probability  $\rho = 30\%$ .

<b>The average</b> ( $\alpha$ )	<b>Sweep method</b> (sec)	<b>Davidson</b> (sec)	<b>Jacobi-Davidson</b> (sec)
0.01	0.189604	0.221786	0.22305
0.02	0.380996	0.542062	0.547262
0.03	0.548002	0.855689	0.864358
0.04	0.611762	1.020124	1.029784
0.05	0.641472	1.136819	1.140071

**Table B.4** The average time for the probability  $\rho = 40\%$ .

<b>The average</b> ( $\alpha$ )	<b>Sweep method</b> (sec)	<b>Davidson</b> (sec)	<b>Jacobi-Davidson</b> (sec)
0.01	0.292798	0.338709	0.340813
0.02	0.589781	0.842349	0.846137
0.03	0.710656	1.112726	1.123538
0.04	0.784949	1.347352	1.352365
0.05	0.795142	1.396915	1.40682

**Table B.5** The average time for the probability  $\rho = 50\%$ .

<b>The average</b> ( $\alpha$ )	<b>Sweep method</b> (sec)	<b>Davidson</b> (sec)	<b>Jacobi-Davidson</b> (sec)
0.01	0.389864	0.444352	0.445956
0.02	0.801734	1.126114	1.134751
0.03	0.93175	1.463695	1.481317
0.04	0.955312	1.66718	1.678661
0.05	0.953868	1.733344	1.748633

**Table B.6** The pairs (the winning times, the average iterations) derived by the sweep method.

$\alpha$	$\rho$				
	10%	20%	30%	40%	50%
0.01	(573, 5.456)	(682, 6.547)	(693, 7.338)	(696, 8.077)	(653, 8.823)
0.02	(899, 7.57)	(969, 9.946)	(965, 12.833)	(970, 14.939)	(948, 16.287)
0.03	(967, 9.971)	(990, 14.441)	(994, 17.323)	(990, 18.726)	(985, 19.592)
0.04	(988, 12.592)	(998, 17.238)	(996, 19.161)	(995, 20.045)	(995, 20.593)
0.05	(995, 14.637)	(998, 18.84)	(998, 20.09)	(1000, 20.611)	(998, 21.09)

**Table B.7** The pairs (the winning times, the average iterations) derived by Davidson method.

$\alpha$	$\rho$				
	10%	20%	30%	40%	50%
0.01	(217, 11.271)	(198, 13.889)	(142, 15.865)	(149, 17.601)	(156, 19.531)
0.02	(50, 16.559)	(17, 22.77)	(14, 30.769)	(12, 38.066)	(25, 43.083)
0.03	(15, 22.787)	(9, 36.311)	(0, 47.686)	(4, 55.234)	(6, 60.804)
0.04	(9, 30.524)	(1, 47.136)	(1, 57.878)	(4, 64.992)	(1, 70.821)
0.05	(4, 37.469)	(0, 54.924)	(1, 64.413)	(0, 70.213)	(0, 76.375)

**Table B.8** The pairs (the winning times, the average iterations) derived by Jacobi-Davidson method.

$\alpha$	$\rho$				
	10%	20%	30%	40%	50%
0.01	(210, 11.205)	(120, 13.837)	(165, 15.85)	(155, 17.601)	(191, 19.518)
0.02	(51, 16.558)	(14, 22.797)	(21, 30.802)	(18, 38.097)	(27, 43.186)
0.03	(18, 22.82)	(1, 36.364)	(6, 47.805)	(6, 55.242)	(9, 61.008)
0.04	(3, 30.559)	(1, 47.174)	(3, 57.931)	(1, 64.787)	(4, 70.642)
0.05	(1, 37.516)	(2, 54.878)	(1, 64.286)	(0, 70.063)	(2, 76.607)



# Appendix C

## Numerical Results for Large Matrices

**Table C.1** The average time for the probability  $\rho = 10\%$ .

<b>The average</b> ( $\alpha$ )	<b>Sweep method</b> (sec)	<b>Davidson</b> (sec)	<b>Jacobi-Davidson</b> (sec)
0.01	43.096533	74.769833	72.079305
0.0125	51.19984	87.508189	85.684355
0.015	58.467133	98.713169	103.46567
0.0175	83.272124	154.660466	154.72179
0.02	89.375946	157.640172	163.106513

**Table C.2** The average time for the probability  $\rho = 12.5\%$ .

<b>The average</b> ( $\alpha$ )	<b>Sweep method</b> (sec)	<b>Davidson</b> (sec)	<b>Jacobi-Davidson</b> (sec)
0.01	61.834624	97.997804	98.646445
0.0125	82.823576	135.83201	146.78233
0.015	100.787779	161.225316	164.869424
0.0175	126.848448	263.170247	275.815357
0.02	159.84147	300.086794	306.167374

**Table C.3** The average time for the probability  $\rho = 15\%$ .

<b>The average</b> ( $\alpha$ )	<b>Sweep method</b> (sec)	<b>Davidson</b> (sec)	<b>Jacobi-Davidson</b> (sec)
0.01	64.034402	95.04318	98.826696
0.0125	73.534956	123.3267	126.3869
0.015	141.112899	245.788761	241.473291
0.0175	169.726287	320.411985	317.161061
0.02	187.175658	338.999426	329.843054

**Table C.4** The average time for the probability  $\rho = 17.5\%$ .

<b>The average</b> ( $\alpha$ )	<b>Sweep method</b> (sec)	<b>Davidson</b> (sec)	<b>Jacobi-Davidson</b> (sec)
0.01	100.431357	160.755727	167.475867
0.0125	133.102918	209.980443	218.225558
0.015	171.080092	279.142045	295.348738
0.0175	199.352339	355.561821	360.17815
0.02	198.034576	365.092777	372.140057

**Table C.5** The average time for the probability  $\rho = 20\%$ .

<b>The average</b> ( $\alpha$ )	<b>Sweep method</b> (sec)	<b>Davidson</b> (sec)	<b>Jacobi-Davidson</b> (sec)
0.01	109.211465	182.297913	191.092943
0.0125	158.637674	269.461160	277.743598
0.015	172.33437	331.339107	336.010559
0.0175	191.763305	341.984853	349.542765
0.02	161.006742	320.424625	321.654542

**Table C.6** The pairs (the winning times, the average iterations) derived by the sweep method.

$\alpha$	$\rho$				
	10%	12.5%	15%	17.5%	20%
0.01	(83, 6.87)	(92, 7.95)	(82, 8.51)	(94, 9.83)	(94, 11.1)
0.0125	(95, 8.28)	(96, 10.35)	(87, 10.87)	(91, 12.5)	(93, 15.59)
0.015	(94, 9.91)	(95, 12.39)	(95, 15.53)	(88, 17.73)	(94, 19.58)
0.0175	(92, 13.3)	(95, 15.74)	(89, 18.49)	(94, 24.11)	(94, 26.52)
0.02	(93, 15.32)	(96, 19.01)	(94, 22.24)	(93, 27.72)	(97, 29.48)

**Table C.7** The pairs (the winning times, the average iterations) derived by Davidson method.

$\alpha$	$\rho$				
	10%	12.5%	15%	17.5%	20%
0.01	(3, 24.68)	(1, 27.96)	(4, 30.3)	(4, 39.1)	(2, 51.3)
0.0125	(2, 30.12)	(1, 40.3)	(9, 44.42)	(2, 50.72)	(3, 69.26)
0.015	(3, 36.26)	(3, 49.03)	(2, 67.55)	(6, 71.67)	(1, 98.79)
0.0175	(5, 58.84)	(3, 74.7)	(5, 85.64)	(5, 109.93)	(3, 126.4)
0.02	(3, 60.83)	(3, 85)	(5, 100.3)	(3, 134.69)	(1, 154.17)

**Table C.8** The pairs (the winning times, the average iterations) derived by Jacobi-Davidson method.

$\alpha$	$\rho$				
	10%	12.5%	15%	17.5%	20%
0.01	(14, 24.5)	(7, 27.83)	(14, 30.12)	(2, 39.6)	(4, 51.09)
0.0125	(3, 29.85)	(3, 41.05)	(4, 44.27)	(7, 51.61)	(4, 69.23)
0.015	(3, 36.86)	(2, 48.6)	(3, 66.5)	(6, 74.26)	(5, 97.66)
0.0175	(3, 57.73)	(2, 76.04)	(6, 83.9)	(1, 109.14)	(3, 125.7)
0.02	(4, 60.24)	(1, 84.83)	(1, 98.4)	(4, 132.63)	(2, 154.37)