The determination of sparse eigensystems and parallel linear system solvers

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:


Metadata Record: [https://dspace.lboro.ac.uk/2134/32054](https://dspace.lboro.ac.uk/2134/32054)

Publisher: © Javad Shanechi

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 2.5 Generic (CC BY-NC-ND 2.5) licence. Full details of this licence are available at: [http://creativecommons.org/licenses/by-nc-nd/2.5/](http://creativecommons.org/licenses/by-nc-nd/2.5/)

Please cite the published version.
This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.lboro.ac.uk/) under the following Creative Commons Licence conditions.

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
THE DETERMINATION OF SPARSE EIGENSYSTEMS
AND PARALLEL LINEAR SYSTEM SOLVERS

BY

JAVAD SHANECHI, B.Sc., M.Sc.

A Doctoral Thesis
Submitted in Partial Fulfilment of the Requirements
for the Award of Doctor of Philosophy
of Loughborough University of Technology
June, 1980.

Supervisor: PROFESSOR D.J. EVANS, PH.D., D.SC.,
Department of Computer Studies

© by Javad Shanehchi, 1980.
DECLARATION

I declare that the following thesis is a record of research work carried out by me, and that the thesis is of my own composition. I also certify that neither this thesis nor the original work contained therein has been submitted to this or any other institution for a degree.

J. SHANEHCHI
ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Professor D.J. Evans without whose guidance, encouragement and useful ideas, this thesis would never have been completed and to the Department of Computer Studies for financial support.

I am grateful to Mr. S. Bedi for his help in the PDP 11/40 Laboratory. Finally, I would like to thank Miss J.M. Briers who typed this thesis so professionally.
## CONTENTS

### Chapter I: INTRODUCTION

1.1 The Two-Point Boundary Value Problem  
1.2 Vibration of a Mechanical Structure

### Chapter II: BASIC LINEAR ALGEBRAIC THEORY

II.1 Notations and Definitions  
II.2 Eigenvalues and Eigenvectors  
II.3 Some Methods for Obtaining Eigenvalues  
   II.3.1 The Power Method  
   II.3.2 Inverse Iteration  
   II.3.3 Newton-Raphson Iteration, Secant and Muller's Method  
   II.3.4 Sturm Sequences and the Bisection Method  
   II.3.5 Methods Based on the Application of the Rayleigh Quotient

### Chapter III: SPARSE GAUSSIAN ELIMINATION WITH PIVOTAL STRATEGIES AND THE DETERMINATION OF EIGENVALUES

III.1 Introduction  
III.2 Symmetric Gaussian Elimination and Local Minimisation Algorithms  
III.3 Some Pivotal Strategies for Arbitrary Matrices  
III.4 Nested Dissection Algorithms  
III.5 Determination of the Eigenvalues of a Matrix by the Elimination Method

### Chapter IV: NUMERICAL EXPERIMENTS ON ELIMINATION METHODS

IV.1 Introduction  
IV.2 Numerical Examples  
IV.3 Conclusion
Chapter V: ITERATIVE METHODS FOR THE EVALUATION OF THE EIGENVALUES AND EIGENVECTORS OF SYMMETRIC MATRICES

V.1 Introduction 88
V.2 Some Properties of the Rayleigh Quotient 89
V.3 SOR Method for the Minimisation (maximisation) of the Rayleigh Quotient 92
V.4 Determination of the Extreme Eigenvalues by a Method Based on a New Splitting of the Matrix C_i 96
   V.4.1 General Theory 96
   V.4.2 Shift of Origin of the Iteration Matrix 103
   V.4.3 Introduction of the Preconditioning Factor 106
V.5 Determination of Good Estimates of the Parameter \omega 110

Chapter VI: NUMERICAL EXPERIMENTS ON ITERATIVE METHODS

VI.1 Introduction 131
VI.2 The Initialisation of the Starting Vector 132
VI.3 Numerical Examples 134
VI.4 Conclusion and Final Remarks 150

Chapter VII: THE QUADRANT INTERLOCKING FACTORISATION (Q.I.F.) METHOD

VII.1 Introduction 153
VII.2 The Factorisation Process of the Q.I.F. Method 153
   VII.2.1 General Description 153
   VII.2.2 The Computation of the Matrices W and Z 155
   VII.2.3 The Computational Cost of the Factorisation Process 156
VII.3 Solution of the Linear System 159
VII.4 The Error Analysis of the Q.I.F. Method 162
VII.5 The Error Analysis of the Solution Process 169
   VII.5.1 The Error Analysis of the Linear System Wy=b 169
   VII.5.2 The Error Analysis of the Linear System Zx=y 172
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>VII.6 A Modification of the Q.I.F. Method</td>
<td>175</td>
</tr>
<tr>
<td>VII.6.1 General Description</td>
<td>175</td>
</tr>
<tr>
<td>VII.6.2 The Computation of W and Z Matrices</td>
<td>176</td>
</tr>
<tr>
<td>VII.6.3 Solution of the Linear System</td>
<td>177</td>
</tr>
<tr>
<td>Chapter VIII: NEW QUADRANT INTERLOCKING FACTORISATION METHODS</td>
<td></td>
</tr>
<tr>
<td>VIII.1.1 The W.D.Z. Factorisation of the Coefficient Matrix</td>
<td>180</td>
</tr>
<tr>
<td>VIII.1.2 Solution of the Linear Systems</td>
<td></td>
</tr>
<tr>
<td>VIII.2.1 The Gauss-Jordan Form of the Q.I.F. Method</td>
<td></td>
</tr>
<tr>
<td>VIII.2.2 The Evaluation of the R.H.S. Vector and the Solution of the Linear System</td>
<td></td>
</tr>
<tr>
<td>VIII.3.1 The Choleski Form of the Q.I.F. Method</td>
<td></td>
</tr>
<tr>
<td>VIII.3.2 The Computation of the matrix W</td>
<td></td>
</tr>
<tr>
<td>VIII.3.3 Solution of the Linear System</td>
<td></td>
</tr>
<tr>
<td>VIII.4 The Relationship Between the Choleski Form of the Q.I.F. Method</td>
<td></td>
</tr>
<tr>
<td>VIII.5 The Error Analysis of the Choleski Form of the Q.I.F. Method</td>
<td></td>
</tr>
<tr>
<td>REFERENCES</td>
<td>202</td>
</tr>
<tr>
<td>APPENDIX 1: Fortran Programs on Elimination Methods for the Solution of Eigensystems</td>
<td>212</td>
</tr>
<tr>
<td>APPENDIX 2: Fortran Programs on Iterative Methods for the Solution of Eigensystems</td>
<td>245</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION
The algebraic eigenvalue problem occurring in a variety of problems in the Natural, Engineering and Social Sciences areas can with some advantage be solved by matrix methods. However, these problems become more difficult to handle when the matrices involved are large and sparse because the storage and manipulations of these types of matrices have to be achieved in such ways that firstly, no storage is wasted by retaining the zero elements of the matrix and secondly, saving valuable computer time by not operating on the zero elements when unnecessary. For this purpose, we have previously developed a software package on the storage and manipulation of sparse matrices, SHANEHCHI [5], which consists of basic matrix operations (i.e. addition, multiplication, etc.) and the solution of linear systems by iterative methods. However, in that work we encountered a great deal of difficulty in handling the operations which generate non-zero elements during processes such as the Gaussian elimination process.

In the first part of the thesis, we concentrate on sparse eigenvalue problems.

There are two general strategies which are used in methods for the determination of eigenvalues and eigenvectors of a matrix. The first type are methods which change the structure of the original matrix and therefore, may generate new non-zero elements during the process and hence destroy sparsity i.e. transformation methods and elimination methods. These methods are almost always preferable when all the eigenvalues and/or eigenvectors of a large sparse matrix are required and also often when a partial eigensolution is required.

The second type consists of methods which are iterative and do not change the structure of the original matrix and therefore maintain the sparsity pattern of the matrix and are often used when only one or several
of the highest or lowest eigenvalues and/or their corresponding eigenvectors are required.

In the third and fourth chapters we set out to evaluate the eigenvalues and eigenvectors of large sparse matrices by the application of sparse Gaussian elimination techniques as an intermediate process. Pivotal strategies are studies which enable us to generate as few non-zero elements as possible while still maintaining accuracy.

In the fifth and sixth chapters we turn our attention to iterative methods for which the problem of fill-ins does not occur. We introduce a new preconditioned iterative method for evaluating the largest and smallest eigenvalues and their corresponding eigenvectors of large sparse symmetric matrices. The method is compared with S.O.R. and shows that in most cases the overall computational work required is less than that required by S.O.R.

We commence by giving a few examples of large sparse eigenvalue problems.

I.1 THE TWO-POINT BOUNDARY VALUE PROBLEM

\[-y'' - \lambda y = 0\]  \hspace{1cm} (1.1.1)

under the boundary conditions,

\[y(0) = y(1) = 0,\]

which is the prototype of general problem called the Sturm-Liouville problem (TODD [1]) which was studied by J.C.F. Sturm and J. Liouville.

The above problem occurs when we separate the variables in the wave equation:

\[\frac{a^2 z}{\omega^2} = \frac{a^2 z}{\omega^2} \]

\[\frac{\partial^2 z}{\partial x^2} = \frac{\partial^2 z}{\partial t^2} \]
by assuming a solution of the form

\[ z(x,t) = y(x) \exp i\sqrt{\lambda} t. \]

The problem can be interpreted in terms of the vibration of a uniform string with fixed end-points \((x=0, x=1)\) and initial displacement \(z(x,0)\) if the boundary conditions are as follows:

\[ z(0,t) = z(1,t) = 0 \]

and the \(z(x,0)\) are given.

It can be seen that the above problem \((1.1.1)\) has a non-trivial solution if and only if \(\lambda\) is an eigenvalue of the continuous problem, in this case

\[ \lambda = (r\pi)^2 , \quad r=1,2,... \quad (1.1.2) \]

with the corresponding characteristic functions:

\[ y(x) = \sin r\pi x. \]

The two methods suggested by TODD [1], to solve this problem both yield a sparse matrix equation. For example, the first method uses a natural finite difference discretisation of \((1.1.1)\) as follows. Find a non-zero vector

\[ Y = [y_0, y_1, \ldots, y_n, y_{n+1}=0]^T \]

such that

\[ \frac{(y_{r+1}-2y_r+y_{r-1})}{h^2} - \lambda y_r = 0, \quad r=1,2,...,n \]

where \(\lambda\) denotes the eigenvalue of the discretised problem. As \(r\) differs from 1 to \(n\), we have \(n\) equations which can be combined to give the matrix equation,

\[
\begin{bmatrix}
2 & -1 & 0 & 0 & \ldots & 0 \\
-1 & 2 & -1 & & & \\
0 & -1 & 2 & -1 & & \\
& & \ddots & \ddots & \ddots & \\
& & & 0 & -1 & 2 \\
& & & & -1 & 2
\end{bmatrix}
\begin{bmatrix}
\hat{y}
\end{bmatrix}
= \lambda h^2 \begin{bmatrix}
\hat{y}
\end{bmatrix}
\]

where \(\hat{y} = [y_1, y_2, \ldots, y_{n-1}, y_n]^T\).
For the simple case of equal intervals, the solution is well known and is given as follows:
\[ \Lambda = 4n^{-2} \sin^2 \frac{\pi r}{2(n+1)}, \quad r=1,2,...,n \]
\[ = 4(n+1)^2 \sin^2 \frac{\pi r}{2(n+1)}; \]
however, for unequal intervals, the simple analytical solution is no longer valid and sparse matrix numerical techniques have to be employed.

For the second example, we combine the method of Galerkin (1871-1945) with recent developments in the theory of approximation and seek a numerical approximation to the solution of the boundary value problem,
\[ -y''-\lambda q(x)y = 0 \quad (1.1.3) \]
with conditions,
\[ y(0) = y(1) = 0 \]
in the form,
\[ w(x) = w_\beta(x) = \sum_{i=1}^{n} \beta_i b_i(x); \quad (1.1.4) \]
where \( b_i(x), i=1,2,...,n \) are linearly independent functions and \( \beta \) is a vector of constants and because the problem is homogeneous, we can normalise \( \beta \) such that \( ||\beta||_2 \) is taken to be unity. Note that the above solution is, in fact, an approximation to the integral equation corresponding to the solution of (1.1.3). See POGORZELESKI [38].

Since we are, in general, unable to select the vector \( \beta \) and the constant \( \lambda \) to ensure that the residual of \( w(x) \), i.e.,
\[ w''(x)+\lambda q(x)w(x) \]
is small, we ask that this value be orthogonal to all the basis functions \( b_j(x) \), i.e.,
\[ \int_{0}^{1} [w''(x)+\lambda q(x)w(x)]b_j(x)dx = 0, \quad i=1,2,...,n. \quad (1.1.5) \]
The above expression can be rewritten as:
\[ \int_{0}^{1} w''(x)b_i(x)dx + \int_{0}^{1} \lambda q(x)w(x)b_i(x)dx = 0, \quad i=1,2,...,n. \quad (1.1.6) \]
After integration by parts, where
\[ du = w''(x) \, dx \]

and \[ v = b\, \! (x) \]

we have:

\[ w'(x)b\, \! (x) \left| \int_{0}^{1} - \int_{0}^{1} w'(x)b\, \! (x) \, dx + \int_{0}^{1} \lambda q(x)w(x) \, dx = 0, \quad i = 1, 2, \ldots, n. \quad (I.1.7) \]

Assuming that \( b\, \! (0) = b\, \! (1) = 0 \) and substituting for \( w(x) \) its equivalent value as given in (I.1.4) we have:

\[ - \int_{0}^{1} \sum_{j} \beta_{j}b_{j}(x)b\, \! (x) \, dx + \int_{0}^{1} \sum_{j} \beta_{j}b_{j}(x)\lambda q(x)b\, \! (x) \, dx \]

\[ = - \sum_{j} \int_{0}^{1} \beta_{j}b_{j}(x)b\, \! (x) \, dx + \sum_{j} \int_{0}^{1} \lambda q(x)b_{j}(x)b\, \! (x) \, dx = 0 \quad (I.1.8) \]

for \( i = 1, 2, \ldots, n. \)

If we define

\[ a\, \! _{i,j} = \int_{0}^{1} b_{i}(x)b_{j}(x) \, dx \]

and

\[ b\, \! _{i,j} = \int_{0}^{1} q(x)b_{i}(x)b_{j}(x) \, dx \]

we have:

\[ - \sum_{j} \beta_{j}a\, \! _{i,j} + \lambda \sum_{j} \beta_{j}b\, \! _{i,j} = 0, \quad i = 1, 2, \ldots, n, \]

and now the problem is to determine a vector \( \beta \) normalised such that \( ||\beta||_{2} = 1 \) and a constant \( \lambda \) such that

\[ A\beta = \lambda B\beta \quad (I.1.9) \]

where

\[ A = [a\, \! _{i,j}] \]

and

\[ B = [b\, \! _{i,j}], \quad i, j = 1, 2, 3, \ldots, n, \]

which is the generalised eigenvalue problem and a basis function is defined in order to ensure that the matrices \( A \) and \( B \) are sparse. Such a basis function can be obtained by choosing the \( b_{i}(x), \quad r = 1, 2, \ldots, n \) to be the pyramid functions as indicated:
In this case, both the matrices $A$ and $B$ will be tridiagonal no matter what $q(x)$ is. By choosing $q(x)=1$, it can be seen (TODD [1]) that matrix $A$ and $B$ are of the simple forms as follows:

$$A = (n+1) \begin{bmatrix} 2 & -1 & & & & \vdots & & & \end{bmatrix}_{n \times n}$$

and

$$B = \frac{1}{6(n+1)} \begin{bmatrix} 4 & 1 & & & & \vdots & & & \end{bmatrix}_{n \times n}$$
I.2 VIBRATION OF A MECHANICAL STRUCTURE

Another classical example of an eigenvalue problem is given by a mechanical system oscillating around a point of equilibrium. The relation
\[ mx'' + kx = 0 \]  \hspace{2cm} (I.2.1)
gives us the balancing static and kinetic energy of a point of mass \( m \), fastened elastically at the origin with a stiffness (spring constant) \( k \). It is well known that this equation has periodic solutions of the form:
\[ x = x_0 \sin \omega t \]
with the eigenvalue \( \omega^2 \) determined as \( \omega^2 = k/m \).

With a mass matrix \( M \) and a stiffness matrix \( k \) we can likewise describe a large interconnected mechanical system with the equation
\[ M\ddot{x} + kx = 0 \]  \hspace{2cm} (I.2.2)
where \( \dot{x} \) is now a vector.

If all the components are assumed to oscillate at the same frequency
\[ x = y \sin \omega t \]
the equation (I.2.2) can be rewritten as
\[ -M\omega^2 y + ky = 0 . \]  \hspace{2cm} (I.2.3)
Replacing \( \lambda \) for \( \omega^2 \) the equation (I.2.3) becomes
\[ (k - \lambda M)y = 0 , \]
which is a generalised eigenvalue problem where in all realistic cases, \( k \) and \( M \) are symmetric with \( M \) a positive definite and diagonal matrix in simple cases whilst \( k \) is positive semi-definite. The systems which are continuous in space, such as membrane to elastic bodies can be reduced by methods of finite difference or finite elements, giving stiffness and mass matrices or a very regular shape where the same description can be applied to them.

An example, shown in [2] illustrates the structure of the matrices \( k \) and \( M \) when we apply the method of finite elements with first order rectangular elements to an L-shaped membrane.
By applying the following computational stencil

\[ k = \begin{bmatrix}
  1 & 1 & 1 \\
  1 & -8 & 1 \\
  1 & 1 & 1
\end{bmatrix}, \quad M = \begin{bmatrix}
  4 & 16 & 4 \\
  4 & 1 & 16 \\
  1 & 4 & 4
\end{bmatrix} \]

at each point of the grid of the following L-shaped region,

we obtain the generalised eigenvalue problem \( ku = \lambda Mu \) where:

\[
k = -\frac{1}{3h} \begin{bmatrix}
-8 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & -8 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & -8 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 1 & -8 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & -8 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & -8 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & -8 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -8
\end{bmatrix}
\]

and

\[
M = \frac{h}{36} \begin{bmatrix}
16 & 4 & 4 & 1 & 0 & 0 & 0 & 0 \\
4 & 16 & 4 & 1 & 0 & 0 & 0 & 0 \\
4 & 1 & 16 & 4 & 1 & 0 & 0 & 0 \\
1 & 4 & 16 & 4 & 1 & 0 & 0 & 0 \\
0 & 0 & 4 & 1 & 16 & 4 & 0 & 0 \\
0 & 0 & 1 & 4 & 4 & 16 & 4 & 0 \\
0 & 0 & 0 & 1 & 0 & 4 & 16 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 4 & 16
\end{bmatrix}
\]
In the second part of this thesis we discuss the subject of parallel computation. Our work will be the extension of a factorisation method first introduced by EVANS & HATZOPoulos [14] of a matrix in which interlocking matrix quadrant factors are considered in lieu of the standard LU triangular factors. New methods will be presented with their relative error and computational cost analysis. We give a concise background description here of the general classes of parallel computers.

The basic classifications made by FLYNN [13] define two general classes of parallel computers which are Single Instruction Stream Multiple Data Stream (SIMD) and Multiple Instruction Stream Multiple Data Stream (MIMD) computers. See also STONE [11].

The SIMD type of parallel computer consists of an array of processors. Each of these processors execute the same string of instructions on different data. Figure I.1 illustrates a P processor SIMD computer.
Note that, unlike a sequential computer, each of the processors in a SIMD computer are unable to generate their own instructions and the instructions are provided by a control unit which is usually a computer itself. The data streams provide the data for each of the processors from private memories each of which is associated with a processor. Each processor executes the same instruction generated by the control unit on its own data simultaneously.

The MIMD computer or multiprocessor is basically a minicomputer network. Unlike the SIMD computer, each processor of an MIMD computer generates its own instruction stream which it executes on its own data stream. The figure (1.2) illustrates an MIMD computer with p processors.

![Diagram of MIMD computer](image-url)

**FIGURE (1.2)**
As shown in figure (1.2), each processor has its own control unit and so is able to generate its own instruction stream. Therefore, it is possible to execute different instructions simultaneously. The processors need not be identical because they are independent of one another, however, they are compatible with each other. Each processor has its own data stream which is obtained from two sources. A large primary memory and a private memory associated with each processor.

For a full description of SIMD and MIMD and other parallel computers (see DUNBAR [16] and FLYNN [13]).

Generally, a parallel computer with P identical processors is potentially P times as fast as a single computer. However, this limit can only be achieved in idealized situations where the following assumptions made by KUCK [15] hold:

a) Any number of processors and memories may be used at any time.

b) Each processor may perform any of the basic arithmetic operations (i.e. addition, subtraction, multiplication and division) at any time, but different processors may perform different operations at the same time.

c) No time is required to communicate data between the processors and memories.

d) Instructions are always available for execution as required and are never held up by a control unit.

e) There are never any accessing conflicts in the memory.

f) Each operation takes the same amount of time, which will be referred to as a unit step.

Indeed the above assumptions are our essential criteria in the computational costs of the methods which we discuss later. Also these methods will be applicable to parallel computers of the SIMD and MIMD type.
CHAPTER II

BASIC LINEAR ALGEBRAIC THEORY
II.1 NOTATIONS AND DEFINITIONS

In this thesis, a matrix will always be denoted by a capital letter, e.g., A, Z, W, Q. In addition, all the matrices used are square matrices of order n. The elements of a matrix A (say) are denoted by $a_{i,j}$, where $i$ denotes its position in the $i^{th}$ row and $j$ its position in the $j^{th}$ column.

A matrix is said to be sparse if only a small percentage of its elements are non-zero. In a practical sense, a matrix of order $n$ is taken as a sparse matrix, if it has $O(n)$ non-zero elements where $O(n)$ denotes 'the order of $n$', TEWARSON [4]. A sparse matrix can either be ordered or unordered. An ordered sparse matrix has a regular pattern of non-zero or zero elements, for example a tridiagonal, quindiaagonal or a bandwidth matrix. An unordered sparse matrix has its non-zero elements scattered over the matrix array with no regular pattern. The following example is an unordered sparse matrix:

$$
\begin{bmatrix}
  a_{1,1} & 0 & 0 & a_{1,4} & 0 & 0 \\
  0 & a_{2,2} & 0 & 0 & 0 & a_{2,6} \\
  a_{3,1} & 0 & 0 & a_{3,4} & 0 & 0 \\
  a_{4,1} & 0 & a_{4,3} & 0 & 0 & a_{4,6} \\
  a_{5,1} & 0 & 0 & 0 & a_{5,5} & 0 \\
  0 & a_{6,2} & 0 & 0 & a_{6,5} & 0
\end{bmatrix} \quad (II.1.1)
$$

In order to save computer storage, only the non-zero elements of a sparse matrix are stored. This is usually in PACKED FORM. There are many different methods for packed form storage of sparse matrices. These are outlined in SHANEHCHI [5], CURTIS & REID [6] and PAGE & WILSON [7].

In order to make a sparse matrix easier to write, collections of zero elements are usually replaced by a space with a large 0 in it. For example, if matrix W is a $(7\times7)$ matrix of the form:
Then, it would be written for simplicity as:

\[
\begin{bmatrix}
    w_{1,1} & 0 & 0 & 0 & 0 & 0 & w_{1,7} \\
    w_{2,1} & w_{2,2} & 0 & 0 & 0 & w_{2,6} & w_{2,7} \\
    w_{3,1} & w_{3,2} & w_{3,3} & 0 & w_{3,5} & w_{3,6} & w_{3,7} \\
    w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} & w_{4,6} & w_{4,7} \\
    w_{5,1} & w_{5,2} & w_{5,3} & 0 & w_{5,5} & w_{5,6} & w_{5,7} \\
    w_{6,1} & w_{6,2} & 0 & 0 & 0 & w_{6,6} & w_{6,7} \\
    w_{7,1} & 0 & 0 & 0 & 0 & 0 & w_{7,7}
\end{bmatrix}
\]

Likewise if \( A \) is a \((5*5)\) upper triangular matrix thus:

\[
A = \begin{bmatrix}
    a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\
    0 & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\
    0 & 0 & a_{3,3} & a_{3,4} & a_{3,5} \\
    0 & 0 & 0 & a_{4,4} & a_{4,5} \\
    0 & 0 & 0 & 0 & a_{5,5}
\end{bmatrix}
\]

Similarly, this would be written as:

\[
A = \begin{bmatrix}
    a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\
    a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & \text{ } \\
    a_{3,3} & a_{3,4} & a_{3,5} & \text{ } \\
    0 & a_{4,4} & a_{4,5} & \text{ } \\
    a_{5,5} & \text{ } \\
\end{bmatrix}
\]

The determinant of a matrix will be denoted either by \( \text{det}(A) \) or \( |A| \). For the singular matrix \( A \), this value is zero i.e.,

\[
\text{det}(A) = |A| = 0
\]

The expansion of \( \text{det}(A) \) by its \( i \) th row can be defined as:

\[
\text{det}(A) = a_{i,1}a_{i,1} + a_{i,2}a_{i,2} + \ldots + a_{i,n}a_{i,n} = \sum_{k=1}^{n} a_{i,k}a_{i,k}
\]
where
\[ a_{i,j} = (-1)^{i+j} \det(M_{i,j}) , \quad j=1,2,\ldots,n \] (II.1.8)
and \( M_{i,j} \) \((i,j=1,2,\ldots,n)\) is the matrix obtained from \( A \) by removing the elements of its \( i \)th row and \( j \)th column. Clearly, there is a similar expression for the expansion of \( \det(A) \) by its \( j \)th column where we have
\[
\det(A) = a_{1,j} a_{1,j}^1 + a_{2,j} a_{2,j}^2 + \ldots + a_{n,j} a_{n,j}^n = \sum_{k=1}^{n} a_{k,j} a_{k,j}^k
\] (II.1.9)

The following are some of the properties of determinants which are used later in Chapter III.

1) If every element of a row (column) of a square matrix \( A \) is zero, then the matrix is singular, i.e. \( \det(A)=0 \).

2) If matrix \( B \) is obtained from matrix \( A \) by interchanging any two of its rows (columns), then \( \det(B)=-\det(A) \). Therefore, if matrix \( B \) is obtained from \( A \) by applying \( P \) number of such interchanges then \( \det(B)=(-1)^P \det(A) \).

3) If matrix \( B \) is obtained from \( A \) by adding to the elements in the \( i \)th row (column), a scalar multiple of the corresponding elements of another row (column) then \( \det(B)=\det(A) \).

4) If two rows (columns) of a matrix are identical then its determinant is equal to zero.

If we cancel out all but rows \( i_1,i_2,\ldots,i_p \) and columns \( k_1,k_2,\ldots,k_p \) from an \( m\times n \) matrix \( A \), we define the determinant of the resulting matrix to be a minor of \( A \) of order \( P \),
\[
A \left[ \begin{array}{ccc} i_1 & i_2 & \ldots & i_p \\ k_1 & k_2 & \ldots & k_p \end{array} \right] = \det \left[ \begin{array}{cccc} a_{i_1,k_1} & \cdots & a_{i_1,k_p} \\ \vdots & & \vdots \\ a_{i_p,k_1} & \cdots & a_{i_p,k_p} \end{array} \right] \] (II.1.10)
The minors for which \( i_1 = k_1, i_2 = k_2, \ldots, i_p = k_p \) are called the principal minors of \( A_i \), LANCASTER [8].

The matrix \( B \) is said to be the transpose of a matrix \( A \) if

\[
b_{i,j} = a_{j,i} \quad \text{for } i,j = 1,2,\ldots,n.
\]  

(II.1.11)

The matrix \( B \) is usually denoted by \( A^T \) or \( A' \). From the above definition it can be easily seen that if the matrix \( A \) is symmetric (i.e. \( a_{i,j} = a_{j,i} \) for all \( i,j \)) then,

\[
A^T = A.
\]  

(II.1.12)

The matrix \( A \) is said to be orthogonal if

\[
A^T A = I,
\]  

(II.1.13)

where \( I \) is the unit matrix with elements \( I_{i,j} \) defined as follows:

\[
\begin{cases}
I_{i,i} = 1 & , i=1,2,\ldots,n \\
I_{i,j} = 0 & , i,j=1,2,\ldots,n \text{ and } i \neq j.
\end{cases}
\]  

(II.1.14)

The inverse of a square matrix \( A \) (if it exists) is a matrix usually denoted by \( A^{-1} \), if and only if

\[
A A^{-1} = A^{-1} A = I.
\]  

(II.1.15)

Vectors are represented as underlined lower case letters with their elements denoted by the same letter, but with a lower suffix giving the position of that element in the vector. For example, the column vector \( v \) with its elements denoted by \( v_i \) is written as:

\[
v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.
\]  

(II.1.16)

To each column vector as shown above there exists a corresponding row vector which is usually denoted by the same name but with an upper suffix \( T \). The corresponding row vector of the column vector \( v \) in (III.16) is

\[
v^T = [v_1, v_2, \ldots, v_n].
\]  

(II.1.17)
We can now demonstrate another notation for matrices. If $w_i$, $i=1,2,...,n$ is a column vector of the $(n \times n)$ matrix $W$, then we have:

$$W = [w_1, w_2, ..., w_n] \quad \text{(II.1.18)}$$

A permutation matrix is an $n \times n$ matrix of the form

$$P = [e_{i_1}, e_{i_2}, ..., e_{i_n}] \quad \text{(II.1.19)}$$

where $i_1, i_2, ..., i_n$ is a permutation of $1,2,...,n$ and $e_{ij}$ represents a vector with unity in the $j$th position and zero elsewhere. Hence, every permutation matrix has one and only one non-zero element with value 1 in each row and column. Finally, for any permutation matrix $P$ we have

$$P*P^T = P^T*P = I,$$  \hspace{1cm} \text{(II.1.20)}

hence

$$P^T = P^{-1}. \quad \text{(II.1.21)}$$

By the TRACE, $\text{tr}A$, of a matrix $A=(a_{ij})$, $i,j=1,2,...,n$ we mean the sum of the diagonal elements of the matrix:

$$\text{tr}A = \sum_{i=1}^{n} a_{i,i} \quad \text{(II.1.22)}$$
Ordering Vectors and Consistently Ordered Matrices

Definition (II.1.1)

The vector \( \mathbf{y} = (y_1, y_2, \ldots, y_n) \) where \( y_i \) are integer numbers is said to be an ordering vector of a matrix \( A \) of size \( n \) if for any pair of associated integers \( i \) and \( j \) (i.e. the integers \( i \) and \( j \) are associated with respect to \( A \) if \( a_{i,j} \not= 0 \) or \( a_{j,i} \not= 0 \) where \( i \not= j \) we have \( |y_i - y_j| = 1 \).

Definition (II.1.2)

An ordering vector \( \mathbf{y} \), for the matrix \( A \), is a compatible ordering vector for \( A \) if:

i) \( y_i - y_j = 1 \) where \( i \) and \( j \) are associated and \( i > j \),

ii) \( y_i - y_j = -1 \) \( \text{if } i < j \).

Definition (II.1.3)

A matrix \( A \) is said to be consistently ordered if for some \( t \) there exist disjoint subsets \( s_1, s_2, \ldots, s_t \) of \( N = \{1, 2, \ldots, n\} \) such that:

\[
\sum_{k=1}^{t} s_k = N
\]

and such that if \( i \not= j \) and \( i \) and \( j \) are associated, then \( j \in s_{k+1} \) if \( j > i \) and \( j \in s_{k-1} \) if \( j < i \), where \( s_k \) is the subset containing \( i \).

Matrices with Property A

Definition (II.1.4)

A matrix \( A \) of order \( n \) has property A if there exist two disjoint subsets \( s_1 \) and \( s_2 \) of \( N = \{1, 2, \ldots, n\} \) such that:

\[
s_1 \cup s_2 = N
\]

and for any non-zero element \( a_{i,j} \) (or \( a_{j,i} \)) of \( A \) where \( i \not= j \) then \( i \in s_1 \) and \( j \in s_2 \) or else \( i \in s_2 \) and \( j \in s_1 \).
It can be proved, see YOUNG [58], that there exists an ordering vector for a matrix $A$ if and only if $A$ has property $A$.

It can be shown that matrices arising from the solution of the Dirichlet problem for a rectangle and more generally the matrices arising from discrete approximation to a large class of elliptic partial differential equation for general regions have property $A$. For a thorough discussion of consistently ordered matrices, property $A$ and their generalisation see YOUNG [58], VARGA [61] and FRANKEL [57].
II.2 EIGENVALUES AND EIGENVECTORS

Suppose that $A$ is an $(n \times n)$ matrix and $\mathbf{x} \neq \mathbf{0}$ is a vector of order $n$. If the vector $\mathbf{x}$ is such that vector $A\mathbf{x}$ is a multiple of vector $\mathbf{x}$ i.e. there exists a $\lambda$ such that

$$A\mathbf{x} = \lambda \mathbf{x},$$

then such a non-zero vector $\mathbf{x}$ is called the eigenvector of $A$ and $\lambda$ is the corresponding eigenvalue. In other words, the basic algebraic eigenproblem is the determination of $\lambda$ such that (II.2.1) has a non-trivial solution. The above system can then be rewritten as

$$(A - \lambda I)\mathbf{x} = \mathbf{0}.$$  

(II.2.2)

It can be shown that a non-trivial solution to this problem exists if and only if

$$\det(A - \lambda I) = 0.$$  

(II.2.3)

Expanding the determinant of the above equation by the Laplace expansion, we have the following polynomial equation which is called the CHARACTERISTIC POLYNOMIAL of $A$:

$$a_0 + a_1 \lambda + \cdots + a_{n-1} \lambda^{n-1} + (-1)^n \lambda^n = 0.$$  

(II.2.4)

Since the coefficient of $\lambda^n$ is not zero, the above equation has always $n$ roots (complex or real) which are the $n$ eigenvalues of matrix $A$ and there exists at least one non-trivial vector $\mathbf{x}$ corresponding to each eigenvalue for which equation (II.2.2) is satisfied.

Throughout this work we will work with real matrices which have real eigenvalues, however, in the above definitions the elements of $A$ may be complex.

Definition (II.2.1): For any matrix $A$ the function $\mathbf{x}^T A \mathbf{x}$ or,

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j$$

(II.2.5)
is called the quadratic form of A. A matrix is said to be positive definite if its quadratic form is positive for all real vectors $x \neq 0$ [10].

A matrix is symmetric positive definite if and only if all its eigenvalues are positive.

**Eigenvalue Bounds**

There are two theorems due to Gerschgorin which define bounds for the eigenvalues of a matrix which in some ways can be used to determine the interval in which the eigenvalues of a symmetric matrix lie. These theorems are as follows:

I) Every eigenvalue of matrix A (say) lies in at least one of the circular discs with centers $a_{i,i}$ and radii $\sum_{i \neq j} |a_{i,j}|$.

II) If $R$ of these discs form a connected domain which is isolated from the other discs, then there exist exactly $R$ eigenvalues of A within this connected domain.

As a result, if A is a symmetric matrix then we have

$$\min_{i} \{a_{i,i} - \sum_{i \neq j} |a_{i,j}|\} < \lambda < \max_{i} \{a_{i,i} + \sum_{i \neq j} |a_{i,j}|\} \tag{II.2.6}$$

where $\lambda$ is any eigenvalue of A.

It can be shown, GANTMACHER [48], that the sum of all the eigenvalues of a matrix A (say) equals the trace of A, i.e.

$$\sum_{i=1}^{n} \lambda_{i} = \sum_{i=1}^{n} a_{i,i} \tag{II.2.7}$$

**Similar Matrices**

If there exists a non-singular matrix $P$ such that

$$B = P^{-1}AP \tag{II.2.8}$$

then matrices $A$ and $B$ are said to be similar and (II.2.8) defines a similarity transformation and if $B$ is symmetric then $P$ will in general be orthogonal i.e.

$$P^{-1} = P^T \tag{II.2.9}$$
and hence, \[ B = P^T A P \] (II.2.10)

The usefulness of such a transformation is that the eigenvalues of A and B are the same. This can be shown easily for if \[ A\mathbf{x} = \lambda \mathbf{x} \] (II.2.11)
then premultiplying by \( P^{-1} \), we have, \[ P^{-1} A \mathbf{x} = \lambda P^{-1} \mathbf{x} \] (II.2.12)
If \( P^{-1} \mathbf{x} = \mathbf{y} \) then we have:
\[ \mathbf{x} = P \mathbf{y} \] (II.2.13)
and
\[ P^{-1} A P \mathbf{y} = \lambda \mathbf{y} \] (II.2.14)
the relation (II.2.14) shows that the eigenvalues of A and \( P^{-1} A P \) are the same and the relation (II.2.13) holds for their eigenvectors.

If matrix P in (II.2.10) is a unitary matrix, then A and \( P^{-1} A P \) are said to be unitary similar. It can be shown that any square matrix is unitary similar to a triangular matrix with the eigenvalues on the diagonal. More generally, any \( n \times n \) matrix with \( n \) linearly independent eigenvectors is similar to a diagonal matrix with the \( n \) eigenvalues on the diagonal.

The Jordan Canonical Form

A matrix which has multiple eigenvalues is not necessarily similar to a diagonal matrix, therefore it is of interest to know what is the most compact form a matrix can be reduced to, using similarity transformations. In order to illustrate the Jordan Canonical Form which is the most compact form for a general matrix we define a sequence of matrices as follows:
\[ J_1(\lambda) = [\lambda] \] (II.2.15)
and for \( p > 1 \) we have a \( (p^*p) \) matrix, \( J_p[\lambda] \), with an eigenvalue of multiplicity \( p \), of the form:
and only one eigenvector \( \mathbf{x} \) where

\[
\mathbf{x}_i = \begin{cases} 
1, & i=1 \\
0, & i>1 
\end{cases}
\]

then the matrix \( J_p[\lambda] \) is called a simple Jordan submatrix of order \( p \).

Let \( A \) be a matrix of order \( n \) with \( s \) distinct eigenvalues \( \lambda_1, \lambda_2, \ldots, \lambda_s \) of multiplicities \( m_1, m_2, \ldots, m_s \) where,

\[
\sum_{i=1}^{s} m_i = n
\]

then there exists a similarity transformation with matrix \( A \) such that \( P^{-1}AP \) has simple Jordan submatrices \( J_p[\lambda_i] \) isolated along the diagonal with all other elements equal to zero. If there are \( q \) submatrices of order \( p_j \), \( j=1, 2, \ldots, q \) associated with any \( \lambda_i \) then the sum of the orders of these submatrices \( (p_j's) \) associated with \( \lambda_i \) is equal to \( m_i \), i.e.,

\[
\sum_{j=1}^{q} p_j = m_i
\]

The matrix \( P^{-1}AP \) is called the Jordan Canonical form of \( A \) and apart from the ordering of the submatrices along the diagonal, is unique.

If the matrix \( A \) of order \( n \) has \( n \) distinct eigenvalues, the Jordan submatrices are all of order 1 and hence the matrix can be transformed to a diagonal matrix.

A matrix \( B \) of order \( n \) is said to be DEFECTIVE if it has fewer than \( n \) distinct eigenvalues, and fewer than \( n \) linearly independent eigenvectors.
Vector and Matrix Norms

The norm of a vector is denoted by $||\mathbf{x}||$ and defined as:

$$||\mathbf{x}||_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}, \quad p=1,2,\ldots.$$  \hfill (II.2.20)

This single value gives an overall assessment of the size of a vector, analogous to the modulus of a complex number. In (II.2.20) when $p=\infty$ then $||\mathbf{x}||_p$ is interpreted as $\max |x_i|, i=1,2,\ldots,n$ and with $p=2$, $||\mathbf{x}||$ is commonly known as the Euclidean length of a vector.

The norms of a vector as defined above satisfies the following relations:

$$||\mathbf{x}|| > 0 \text{ unless } \mathbf{x} = \mathbf{0}$$  \hfill (II.2.21)

$$||k\mathbf{x}|| = |k| ||\mathbf{x}||, \quad \text{for any complex scalar } k$$  \hfill (II.2.22)

$$||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||.$$

Similarly, the norm of a matrix $A$ is denoted by $||A||$ and satisfies the following relations:

$$||A|| > 0 \text{ unless } A = \mathbf{0}$$  \hfill (II.2.24)

$$||kA|| = |k| ||A||, \quad \text{where } k \text{ is a complex scalar}$$  \hfill (II.2.25)

$$||A+B|| \leq ||A|| + ||B||$$  \hfill (II.2.26)

$$||A^*B|| \leq ||A|| * ||B||.$$  \hfill (II.2.27)

For any vector norm there can be defined a corresponding subordinate matrix norm,

$$||A|| = \max_{\mathbf{x} \neq \mathbf{0}} \frac{||A\mathbf{x}||}{||\mathbf{x}||} = \max_{||\mathbf{x}||=1} ||A\mathbf{x}||.$$  \hfill (II.2.28)

Hence, the subordinate matrix norm satisfies,

$$||A\mathbf{x}|| \leq ||A|| * ||\mathbf{x}||.$$  \hfill (II.2.29)

The matrix norms subordinate to vector norms with $p=2$ and $p=\infty$ are,

$$||A||_2 = \text{(maximum eigenvalue of } A^*A)^{1/2},$$  \hfill (II.2.30)

$$||A||_\infty = \max_{i} \sum_{j=1}^{n} |a_{i,j}|.$$  \hfill (II.2.31)
Normalised Vector

A normalised vector, is a vector multiplied by a scalar in order to keep the element size down without changing its direction. In our work the method used for normalising a vector is described below:

If $\vec{x}$ is an $n$ vector with elements $x_i$, $i=1,2,3,\ldots,n$ then the normalised vector $\vec{x}$ is defined as

$$\vec{x} = \left(\frac{x_1}{s}, \frac{x_2}{s}, \ldots, \frac{x_n}{s}\right),$$

where $s$ denotes the largest element in absolute value of the original vector $\vec{x}$. This method of normalisation ensures that the modulus of every element of the vector is less than 1.
II.3 SOME METHODS FOR OBTAINING EIGENVALUES

There are many methods for determining eigenvalues and/or eigenvectors. The method used will depend on the problem and whether both eigenvalue and eigenvector are required or just the eigenvalue. It also depends on the number of eigenvalues required or their positions in the eigenvalue spectrum. Some methods work with the matrix equation (II.2.1) or directly on the matrix A whereas other methods are concerned with finding the roots of the characteristic equation (II.2.4).

In this section we mention briefly a variety of methods and full details can be obtained from WILKINSON [9].

II.3.1 The Power Method

This method is used to find the real eigenvalue of largest modulus. Suppose that the following relationship holds for the eigenvalues of an \((n \times n)\) matrix \(A\).

\[
|\lambda_1| > |\lambda_2| > |\lambda_3| \geq \cdots \geq |\lambda_n|
\]  

(II.3.1.1)

and that there exist \(n\) linearly independent eigenvectors \(x_1\). Therefore any arbitrary vector \(y^{(0)}\) can be expressed as

\[
y^{(0)} = \sum_{i=1}^{n} a_i x_i.
\]  

(II.3.1.2)

If we successively multiply \(y^{(0)}\) by matrix \(A\), such that

\[
y^{(1)} = Ay^{(0)}
\]  

(II.3.1.3)

\[
y^{(2)} = Ay^{(1)} = \lambda^2 y^{(0)}
\]  

(II.3.1.4)

\[
\vdots
\]

\[
y^{(k)} = Ay^{(k-1)} = \lambda^k y^{(0)}
\]  

(II.3.1.5)

and the fact that if

\[
Ax_1 = \lambda_1 x_1
\]  

(II.3.1.6)

then

\[
A^2 x_1 = \lambda_1 Ax_1 = \lambda_1^2 x_1
\]  

(II.3.1.7)
we have the following:

\[ y(k) = A^k \chi(0) = A^k \sum_{i=1}^{n} a_i x_i = \sum_{i=1}^{n} a_i A^k x_i = \sum_{i=1}^{n} a_i \lambda_i^k x_i \]  

or

\[ y(k) = \lambda_1^k (a_1 x_1 + a_2 (\frac{\lambda_2}{\lambda_1})^k x_2 + \ldots + a_n (\frac{\lambda_n}{\lambda_1})^k x_n) \]

The terms \((\frac{\lambda_i}{\lambda_1})^k\) for \(i=2,3,\ldots,n\) will converge to zero as \(k \to \infty\) so that:

\[ y(k) = \lambda_1^k a_1 x_1 \text{ as } k \to \infty \]

and we have

\[ \frac{y_i^{(k+1)}}{y_i^{(k)}} \to \lambda_1 \text{ as } k \to \infty. \]

It can be seen that the smaller the value of \(|\lambda_2/\lambda_1|\) the faster the convergence. In practice, the vector \(\chi^{(k+1)}\) is computed form the formula

\[ \chi^{(k+1)} = A \chi^{(k)} \]  

and then we find \(p\) such that

\[ |z_p^{(k+1)}| \geq |z_i^{(k+1)}|, i=1,2,\ldots,n \]

\[ q_{k+1} = z_p^{(k+1)} \]

\[ \chi^{(k+1)} = \frac{z^{(k+1)}}{q_{k+1}} \]

and we restart the procedure from (II.3.1.13) for \(k+2\). The process stops when \(q_{k+1}\) and \(q_k\) are the same to some specified accuracy \((\varepsilon)\). Finally, the value of \(q_k\) gives an estimate of \(\lambda_1\) and \(\chi^{(k)}\) an estimate of \(X_1\).

As we mentioned earlier, the rate of convergence of the above method depends essentially on the value of \(|\lambda_2/\lambda_1|\). Since from (II.2.1) we have
and this shows that if \( \lambda \) and \( \mathbf{x} \) are the eigenvalue and the eigenvector of \( A \) respectively, then \( (\lambda - p) \), and \( \mathbf{x} \) are the eigenvalue and eigenvector of the matrix \( (A - pI) \), and that the rate of convergence of matrix \( (A - pI) \) now depends on the value of \( \frac{|\lambda_2 - p|}{|\lambda_1 - p|} \). By a suitable choice of \( p \), it may be possible to make the value of \( \frac{|\lambda_2 - p|}{|\lambda_1 - p|} \) smaller than \( \frac{\lambda_2}{\lambda_1} \), and hence speed up the process. This is known as the shift of origin method. It is possible to choose \( p \) such that the largest eigenvalue (in absolute term) of \( (A - pI) \) corresponds to the smallest eigenvalue of \( A \).

II.3.2 Inverse Iteration

If \( \lambda \) and \( \mathbf{x} \) are the eigenvalue and eigenvector of a matrix \( A \) respectively then we have

\[
\begin{align*}
A\mathbf{x} &= \lambda \mathbf{x} \quad \text{(II.3.2.1)} \\
\mathbf{x} &= \lambda A^{-1} \mathbf{x} \\
A^{-1} \mathbf{x} &= \frac{1}{\lambda} \mathbf{x} \quad \text{(II.3.2.3)}
\end{align*}
\]

Equation (II.3.2.3) shows that \( 1/\lambda \) and \( \mathbf{x} \) are the eigenvalue and eigenvector of \( A^{-1} \) (the inverse of \( A \)). Since the calculation of \( A^{-1} \) should be avoided, the system (II.3.2.3) is solved as an iterative procedure defined by

\[
\begin{align*}
\text{i)} & \quad A\mathbf{y}^{(i)} = \mathbf{y}^{(i-1)} \\
\text{ii)} & \quad \mathbf{z}^{(i)} = \frac{\mathbf{y}^{(i)}}{||\mathbf{y}^{(i)}||_2} \quad \text{(II.3.2.4)}
\end{align*}
\]

The system (i) in (II.3.2.4) represents a linear system, where the Gauss-elimination method as fully described in the next chapter in the case of a sparse matrix \( A \) can be used to solve the system. The LU decomposition of \( A \) can also be used to solve the system (i) in (II.3.2.4). Note that this system is solved with the same coefficient matrix \( A \) but with different right hand side vectors and hence the LU decomposition or Gauss-elimination process of \( A \) need only be carried
out once making great savings in time during the process. Therefore the process as implemented by WILKINSON [9] becomes

\[ \begin{aligned}
L \tilde{y} &= \tilde{z}^{(i-1)} \\
U \tilde{y}^{(i)} &= \nu
\end{aligned} \]  

(II.3.2.5)

where

\[ A = L \cdot U \]  

(II.3.2.6)

The first step is not carried out for \( i=1 \) and the second step is replaced by

\[ \tilde{y}^{(1)} = e \]  

(II.3.2.7)

where \( e \) is the vector whose elements are all 1. This starting procedure works well under most conditions.

The smallest eigenvalue of \( A \), after the convergence of (II.3.2.4) is finally given by

\[ \lambda = 1/q_k \]  

(II.3.2.8)

where

\[ q_k = ||\tilde{y}^{(k)}||_2 \]  

(II.3.2.9)

and the eigenvector corresponding to this value will be \( \tilde{y}^{(k)} \).

Though inverse iteration is more difficult to apply than a direct shift of origin for finding the smallest eigenvalue, the rate of convergence is generally greater.

**II.3.3 Newton-Raphson Iteration, Secant and Muller's Methods**

The methods discussed so far, have all been concerned with evaluating a particular eigenvalue of a matrix, e.g., the largest eigenvalue or the nearest eigenvalue to a specified value, etc.

The methods which are described in this section enable us to evaluate the eigenvalues of general matrices and are all root finding methods for determining the zeros of a polynomial. The application of the methods for evaluating the eigenvalues of a matrix is to find the roots of the characteristic equation, shown in (II.2), by working with the determinant of the matrix.

*Using a shift of origin with the inverse iteration method.*
From now on we shall denote the value of the characteristics equation at point \( \lambda \), by \( p_n(\lambda) \) which is zero when \( \lambda \) is the eigenvalue of an \((n \times n)\) matrix \( A \). Therefore, we have,

\[
p_n(\lambda) = \det(A-\lambda I),
\]

and

\[
p_n'(\lambda) = \frac{d}{d\lambda} \det(A-\lambda I).
\]

The Newton-Raphson iteration, as the first method, can be defined as,

\[
\lambda^{i+1} = \lambda^i - \frac{p_n(\lambda^i)}{p_n'(\lambda^i)}, \quad p_n'(\lambda^i) \neq 0.
\]

The process starts with an initial guess, \( \lambda^1 \) at an eigenvalue and is terminated when for some specified accuracy \( \epsilon \), we reach a value \( k \) such that:

\[
|\lambda^{k+1} - \lambda^k| < \epsilon,
\]

the method converges quadratically when applied to ordinary polynomial and can also be applied for matrices which have complex eigenvalues and locate the roots; but to do this the starting guess, i.e., \( \lambda^1 \), must be complex.

One of the disadvantages of this method is the necessity to evaluate the derivative of \( p_n(\lambda) \). For some matrices, \( p_n'(\lambda) \) may not be explicitly known, and even when it is possible to evaluate \( p_n'(\lambda) \) this may require a considerable computational effort. Therefore, in the cases where the determination of \( p_n'(\lambda) \) is impracticable, the secant method can be used instead.

The secant method, as a variant of the Newton-Raphson iteration, uses the same iterative formula as (II.3.3.3) with \( p_n'(\lambda) \) replaced by its finite difference approximation, i.e.,

\[
p_n'(\lambda^i) = \frac{p_n(\lambda^i) - p_n(\lambda^{i-1})}{\lambda^i - \lambda^{i-1}}.
\]

and hence (II.3.3.3) can be rewritten as:
The secant method as described above requires only one determinant evaluation per step, whereas Newton-Raphson method requires the evaluation of the determinant and its differential at each step. On the other hand, the secant method does not converge quite as fast as Newton's method. It can be shown, ANDERSON [20], that if \( \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n \), then starting with \( \lambda^0 < \lambda^1 < \lambda_1 \), or \( \lambda^1 < \lambda^0 < \lambda_n \), the secant method converges monotonically to \( \lambda_1 \) or \( \lambda_n \), respectively.

A common disadvantage of both, the secant method and the Newton-Raphson iteration is that the starting point(s), i.e. \( \lambda^1 \) in the Newton-Raphson or \( \lambda^0, \lambda^1 \) in the secant method, must be close enough to the root \( \lambda \) of \( p_n(x) \) to ensure convergence.

The third method we describe in this section is Muller's method, MULLER [17], which may be used to find any prescribed number of zeros of the characteristic equation, \( p_n(\lambda) \), real or complex. This method is also iterative and converges quadratically in the vicinity of a root, based on computed values of \( p_n(\lambda) \) and not on values of any of the derivatives and obtains both real and complex roots even when these roots are not simple. This method is described in MULLER [17] and FROBERG [18], but briefly in this method, the next approximation \( \lambda^{i+1} \) to an eigenvalue is determined from the three previous approximations, \( \lambda^i, \lambda^{i-1}, \lambda^{i-2} \), where a parabola passes through the points \((\lambda^i, f_i(\lambda^i)), (\lambda^{i-1}, f_{i-1}(\lambda^{i-1})), (\lambda^{i-2}, f_{i-2}(\lambda^{i-2}))\) where

\[
f_j(\lambda^j) = p_n(\lambda^j)
\]  

Therefore if,

\[ h_i = \lambda^i - \lambda^{i-1}, \quad \psi_i = h_i/h_{i-1}, \quad \delta_i = 1 + \psi_i \]

then we have:

\[
\lambda^{k+1} = \lambda^k + \psi_{k+1} h_k
\]
where
\[ \psi_{k+1} = \frac{-2f_k \delta_k}{(g_k^2 + 4f_k \delta_k \psi_k f_{k-1}^2 + f_k^2)} \] (II.3.3.10)
and
\[ g_k = f_{k-2} \psi_{k-1}^2 + f_k^2 (\psi_k + \delta_k) \] (II.3.3.11)

The sign before the radical in (II.3.3.10) is selected such that the modulus of the denominator in the same formula is as large as possible. The process is continued by using (II.3.3.11, II.3.3.10, II.3.3.9 respectively) and \( \lambda_1, \lambda_2, \lambda_3 \) as the three basic approximations.

Note that if the eigenvalue being sought is real, we may encounter complex approximations because of formula (II.3.3.10). However, the complex component in such cases will normally be so small in magnitude that it can be neglected, see [19]. In our numerical experiments in Chapter IV, when evaluating a real eigenvalue, any complex components produced by (II.3.3.10) are suppressed.

With all the three methods (i.e. Newton-Raphson, secant method and Muller’s method) described above, once an eigenvalue has been found steps must be taken to avoid re-determining the same eigenvalue. This is achieved by dividing \( p_n(\lambda) \) by the product of all the differences of the current guess from each eigenvalue previously determined, i.e. if the \( r \) eigenvalues \( \lambda_1, \lambda_2, \ldots, \lambda_r \) have already been evaluated the value \( q \) defined as
\[ q = p_n(\lambda^i) / \prod_{j=1}^{r} (\lambda_j - \lambda^i) \] (II.3.3.12)
can be substituted for \( p_n(\lambda^i) \) in (II.3.3.7), (II.3.3.6) and (II.3.3.3).

II.3.4 Sturm Sequences and The Bisection Method

Suppose that \( A \) is a real symmetric matrix, i.e.
\[ a_{i,j} = a_{j,i}, \quad j,i=1,2,\ldots,n \] (II.3.4.1)
then \( p_x(\lambda) \), the leading principal minor of order \( r \) of the matrix \( (A-\lambda I) \), as defined in (II.1), is given by:
The sequence \( p_n(\lambda) \), \( i=1,2,\ldots,n \) where \( p_0(\lambda) \) is defined to be 1 is a sequence of polynomials in \( \lambda \) where the zeros of \( p_r(\lambda) \) are the eigenvalues of the leading principal submatrix of order \( r \) of \( A \). Hence the roots of \( p_n(\lambda) \) are the eigenvalues of \( A \). It can also be shown that the zeros of \( p_r(\lambda) \) strictly separate those of \( p_{r-1}(\lambda) \), WILKINSON [9].

For a given value \( \lambda=q \), the sequence \( p_i(q) \), \( i=1,2,\ldots,n \) as defined above forms a Sturm sequence. The important property which is used to locate the roots of the characteristic equation \( p_n(\lambda) \) is that:

The number of sign agreements (disagreements) in the sequence \( p_0(q)=1, p_1(q),\ldots,p_n(q) \) equals the number of eigenvalues of \( A \) which are strictly greater (smaller) than \( q \).

Note that if any \( p_i(q) \) is zero, then the sign of \( p_i(q) \) is taken to be the same as \( p_{i-1}(q) \).

The evaluation of the Sturm sequence(s) enables us to obtain successive upper and lower bounds on the roots of the polynomial equation \( p_n(\lambda) \). The operation is called the Bisection method and can be used to compute all the eigenvalues of \( A \) or any particular one.

For example, to obtain the \( k^{th} \) largest eigenvalue \( (\lambda_k) \) of the matrix \( A \) we have the following:

Suppose that \( a \) and \( b \) are the lower and upper bounds for the eigenvalues of matrix \( A \) respectively which can be obtained as shown in (II.2.6), therefore we have,

\[ a < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n < b \quad (II.3.4.3) \]
where $\lambda_i$, $i=1,2,\ldots,n$ denotes the eigenvalues of $A$. Then, the Sturm sequence is evaluated for $q_1=(a+b)/2$. If the number of the agreements in sign in the evaluated sequence is greater than or equal to $(n-k+1)$ then $\lambda_k$ lies in the closed interval $[q_1,b]$, otherwise it lies in the closed interval $[a,q_1]$. The process is now repeated for $q_2$ equal to either $(q_1+b)/2$ if the interval $[q_1,b]$ is used, or $(q_1+a)/2$ if the interval $[a,q_1]$ is chosen. We repeat the process always choosing the interval in which $\lambda_k$ lies until the width of the interval is smaller than some predefined accuracy, $\varepsilon$.

The Bisection method converges with an asymptotic convergence factor of at least $1/2$ and thus is not as fast as the previous methods. However, it is the ability to locate an eigenvalue independently of any others that makes the method so powerful and useful. The eigenvalues obtained by the Bisection method can efficiently be used as the initial guess(es) for a more rapidly converging iteration to obtain further results of greater accuracy.
II.3.5 Methods Based on the Application of the Rayleigh Quotient

The Rayleigh quotient of a matrix $A$ for a non-trivial vector $x$ is defined as:

$$\frac{x^t A x}{x^t x}.$$  \hspace{1cm} (II.3.5.1)

The main property of the Rayleigh quotient of a matrix $A$ is that if the equation $A x = \lambda x$, $(x \neq 0)$ holds, then $\lambda$ is given by the formula

$$\lambda = u(x), \text{ where}$$

$$u(x) = \frac{x^t A x}{||x||^2}. \hspace{1cm} (II.3.5.2)$$

If the matrix $A$ is real and symmetric, then the maximum of $u(x)$ is the largest eigenvalue and the minimum of $u(x)$ gives the smallest eigenvalue of $A$. Therefore, these extreme eigenvalues can be obtained by the maximisation of $u(x)$ in (II.3.5.2) for the largest eigenvalue and by the minimisation of $u(x)$ for the smallest eigenvalue.

There are many minimisation methods which use the above property of the Rayleigh quotient of a symmetric matrix together with a deflation process for obtaining a partial eigensolution. In this section, we describe some of these methods.

The Gradient Method

The Gradient method proposed by Hestenes [52-54] as the first minimisation method described here, defines the direction of the steepest ascent as:

$$\xi = A x - u(x) x.$$  \hspace{1cm} (II.3.5.3)

which is called the gradient of $u(x)$ defined in (II.3.5.2) at $x$.

By the definition of the eigenvalue and eigenvector of a matrix $A$ it is clear that a non-trivial vector $x$ is an eigenvector of $A$ if and only if $\xi = 0$ and therefore the magnitude of $\xi$ related to the magnitude of $x$ can
be taken as a measure of the deviation of \( x \) from an eigenvector.

In the Gradient method, the maximum of \( u(x) \) is obtained by the following iterative formula,

\[
X_{i+1} = X_i + \alpha_i \xi_i \quad i=1,2,\ldots
\]

where \( \xi_i \) is the gradient of \( u(x) \) at \( X_i \). It can be shown that if \( \alpha_i \) is chosen such that

\[
0 < \delta < \alpha_i \leq \frac{2}{M} - \delta
\]

where

\[
M = \lambda_{\text{max}} - \lambda_{\text{min}}
\]

then,

\[
u(x_{i+1}) - u(x_i) \geq c ||\xi_i||_2
\]

where \( c \) is a positive number independent of \( i \). If the starting vector \( X_0 \) is not orthogonal to the eigenvector corresponding to the largest eigenvalue of \( A \) and that \( \alpha_i \) satisfies (II.3.5.5) then the \( u(x_i) \) will converge to the largest eigenvalue of \( A \) whilst the vectors \( x_i \) will converge to the corresponding eigenvector. See HESTENES [52].

To evaluate the smallest eigenvalue it is enough to replace \( \alpha_i \) by \(-\alpha_i\) in (II.3.5.4) where the inequality (II.3.5.7) changes to:

\[
u(x_{i+1}) - u(x_i) \geq c ||\xi_i||_2
\]

Among the three methods of selecting the parameter \( \alpha_i \), the following strategy has been proved to be more efficient.

The choice of the parameter \( \alpha_i \) is made by the formula:

\[
\alpha_i = \beta / ||s_i||
\]

where

\[
s_i = u(\xi_i) - u(x_i)
\]

and \( \beta \) is a fixed number belonging to the interval \((0,1)\). For a thorough discussion on this method see HESTENES [52].
The Conjugate Gradient Method

The conjugate gradient algorithm for the solution of the linear system:

\[ Cx = b \]  \hspace{1cm} (II.3.5.9)

where \( C \) is a positive definite matrix (see LUENBERGER [72], for indefinite matrices) is defined as follows:

\[
\begin{align*}
  &x_0 = 0 \\
  &r_0 = p_0 = b \\
  &\alpha_k = \frac{r_k^T r_k}{p_k^T c p_k} \\
  &x_{k+1} = x_k + \alpha_k p_k \\
  &r_{k+1} = r_k - \alpha_k c p_k \\
  &\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}, \\
  &p_{k+1} = r_{k+1} + \beta_k^p p_k
\end{align*}
\]  \hspace{1cm} (II.3.5.10)

One of the applications of the conjugate gradient algorithm for the evaluation of the eigenvalues and eigenvectors of large sparse symmetric matrices has been developed by RUHE [73] whereby the system:

\[ Au = \lambda_i u_i \]  \hspace{1cm} (II.3.5.11)

is solved by inverse iteration. See WILKINSON [9]. This application can either obtain an accurate eigenvector corresponding to a given eigenvalue \( \lambda \) from the inverse iteration, computing the sequence \( x_1, x_2, \ldots \), from:

\[ (\lambda - A)x_{i+1} = x_i \]  \hspace{1cm} (II.3.5.12)

or obtain both the eigenvalue and the corresponding eigenvector using the Rayleigh quotient. In the first case, where the eigenvalue is known, then the corresponding eigenvector is evaluated from the inverse iteration (II.3.5.12) using the conjugate gradient algorithm (II.3.5.10) with \( C=\lambda I-A \) and \( b=x_0 \) where \( x_0 \) is the starting vector. In the second case, we obtain an estimate of the eigenvalue by evaluating the Rayleigh quotient.
of the vector $x_0$, i.e.

$$\lambda_0 = \frac{x_0^T A x_0}{x_0^T x_0}$$

and then solving the inverse iteration scheme,

$$(\lambda_0 I - A)x_{i+1} = x_i$$

$i = 0, 1, 2, \ldots$ as in the first case. Then, after the convergence of the above system to $x_s$ (say) we determine a better estimate of the eigenvalue by evaluating the Rayleigh quotient of $x_s$:

$$\lambda_1 = \frac{x_s^T A x_s}{x_s^T x_s}$$

and then solving the system:

$$(\lambda_1 I - A)x_{i+1} = x_i$$

$i = s, s+1, \ldots$ by the conjugate gradient algorithm. The process is continued until both the eigenvalue and the corresponding eigenvector have been determined.

Note, however, that the system (II.3.5.12) is very nearly singular and if we evaluate $\alpha_0$ from (II.3.5.10) we get:

$$\alpha_0 = \frac{r_0^T r_0}{p_0^T (\lambda I - A) p_0} = \frac{r_0^T r_0}{r_0^T (\lambda I - A) r_0} = (\lambda - \mu(r_0))^{-1}$$

where

$$\mu(r_0) = \frac{r_0^T A r_0}{r_0^T r_0}$$

and if we use the Rayleigh quotient of the starting vector as the initial value of $\lambda$ then $\alpha_0$ tends to infinity. This however, will affect only the first iteration of the conjugate gradient algorithm provided $\lambda_1$ for which the problem is being solved is separated from the other eigenvalues of $A$.

To overcome this problem we have two choices. The first is to use the special device introduced by LUENBERGER [72] for indefinite matrices where we make a double step using the following algorithm:
\[ u_i = r_0^t c_i r_0, \quad i = 0, 2, 3 \]

\[ p_1 = c r_0 - \frac{u_3}{2u_2} r_0 \]

\[ x_2 = \frac{u_0}{u_2} (c r_0 - \frac{3}{u_2} r_0), \quad (II.3.5.13) \]

\[ r_2 = b - C x_2 \]

and

\[ p_2 = r_2 - \frac{r_2^t c p_1}{u_2} r_0; \]

after that we continue the iteration process as usual using the algorithm (II.3.5.10).

The second choice is to modify the Rayleigh quotient in some systematic manner for obtaining an approximation to \( \lambda_1 \) as:

\[ \lambda = \mu(x) + ||s||_2^2 / d \]

where \( s \) is the initial residual,

\[ s = \mu(x) x - A x \]

and \( \delta \) is a guess of the gap \( \delta \) defined as

\[ \delta = \min_{2 \epsilon \text{sign}} |\lambda - \lambda_1(A)|. \]

For a more detailed description of this method and its practical aspects (see Ruhe [73]).

The Coordinate Relaxation Method

In order to evaluate the minimum of the Rayleigh quotient in the method of Coordinate Relaxation, we start with an arbitrary initial vector \( x \not= 0 \) such that

\[ \mu(x) = \frac{x^t A x}{x^t x} \geq \lambda_1 \]

where \( \lambda_1 \) denotes the smallest eigenvalue of \( A \).

In order to reduce the actual value of \( \mu(x) \), we only change the \( j \)th component of the vector \( x \). That is, we consider the family of new vectors \( x' \) defined as,
\[ x' = \phi x + \gamma e_j \]  

(II.3.5.15)

where \( e_j \) denotes the \( j \)th unit vector and where the scalars \( \phi \) and \( \gamma \) do not vanish simultaneously. The next step is to evaluate the values \( \phi \) and \( \gamma \) such that the Rayleigh quotient of the vector \( x' \) is a minimum, i.e.

\[
\mu(x') = \frac{(\phi x + \gamma e_j)^T A (\phi x + \gamma e_j)}{(\phi x + \gamma e_j)^T (\phi x + \gamma e_j)}
\]

\[
= \frac{\alpha \phi^2 + 2\phi \gamma + \gamma^2}{\beta \phi^2 + 2\phi \gamma + \gamma^2} = \min
\]

(II.3.5.16)

where the following quantities have been used for brevity:

\[
\alpha = \frac{x^T Ax}{x^T x}, \quad \beta = x^T x
\]

\[
f = \frac{e_j^T Ax}{e_j^T x} = u_j, \quad g = x_j
\]

\[
p = \frac{e_j^T Ae_j}{e_j^T e_j}
\]

with \( u_j \) denoting the \( j \)th component of the vector \( u = Ax \).

The problem of minimising (II.3.5.16) leads to the solution of the following system:

\[
\begin{align*}
(a - u^T \beta) \phi + (u_j - u^T x_j) \gamma &= 0 \\
(u_j - u^T x_j) \phi + (a_{j,j} - u^T u') \gamma &= 0
\end{align*}
\]

(II.3.5.18)

which is obtained from differentiating (II.3.5.16) with respect to \( \phi \) and \( \gamma \). The above system can be expressed as:

\[
\begin{bmatrix}
\alpha & u_j \\
u_j & a_{j,j}
\end{bmatrix}
\begin{bmatrix}
\phi \\
\gamma
\end{bmatrix}
= \mu
\begin{bmatrix}
\beta & x_j \\
x_j & 1
\end{bmatrix}
\begin{bmatrix}
\phi \\
\gamma
\end{bmatrix}
\]

which represents a generalised eigenvalue problem. The smallest possible value for \( \mu(x') \) is equal to the smaller of the two eigenvalues \( \mu' \) and the two scalars \( \phi \) and \( \gamma \) are the two components of the corresponding eigenvector.
of (11.3.5.18). Since an eigenvector is evaluated only to within a
multiplicative constant, therefore only one of the components of this
eigenvector can be assumed to be unity provided this component is not zero.
See SCHWARZ [70].

For a systematic decrease of the Rayleigh quotient the vectors e_j are
selected cyclically, and the sequence of n single steps is called a cycle.
A summarised algorithm for the coordinate relaxation method which is given
by SCHWARZ [71] is as follows:

Start: choice of x ≠ 0

U = Ax
α = x^t U, \quad β = x^t x, \quad μ = α/β

cycle: for j = 1, 2, ..., n
f = u_j, \quad g = x_j
p = a_j, j

calculate from (11.3.5.18) u' ≤ u, \quad f = 1 and γ

x_j = x_j + γ
U = U + a_j, \quad \text{where } a_j \text{ is the } j^{th} \text{ column of } A
α = α + 2fγ + pγ^2
β = β + 2gγ + γ^2

μ = α/β

The process is terminated when the Rayleigh quotient μ is numerically
stationary over a complete cycle and when the maximum modulus value of
the γ's is smaller than some specified tolerance ε for a full cycle.

The Lanczos Algorithm

The Lanczos algorithm is another method which uses the evaluation of
the Rayleigh quotient of the coefficient matrix A. In this method,
proposed by Lanczos [74], all the eigenvalues and/or corresponding eigen-
vectors of a symmetric matrix A are evaluated, by first reducing the matrix A into a tridiagonal matrix T which is similar to A, and then computing the eigenvalues of T. For this, the algorithm evaluates an orthogonal matrix Q such that:

\[ Q^T A Q = T. \]  

The eigenvalues of A are therefore, identical to the eigenvalues of T and furthermore, if \( y \) is an eigenvector of T, then \( Qy \) is the corresponding eigenvector of A. The following algorithms recommended by PAIGE [76] ensures a stable form of the Lanczos algorithm whereby the columns of the orthogonal matrix Q are computed sequentially.

We choose a vector \( q \) of norm unity and set \( \beta_1 = 0 \). Then, we compute the following steps for \( i=1,2,...,n \),

1. evaluate the vector \( Aq_i \),
2. set \( q_1 = q_1^TAq_1 \),
3. set \( z = Aq_i - q_1^*$q_1 - \( \beta_i q_i-1 \),
4. set \( \beta_{i+1} = ||z|| \),
5. if \( \beta_{i+1} = 0 \), then stop; otherwise, set \( q_{i+1} = z/\beta_{i+1} \) and continue the process from step 1.

It can be seen from the above algorithm, that:

\[ q_1^Tz = q_1^T(Aq_1 - q_1^TAq_1^*q_1) = \beta_1^T q_1^T - q_1^T(q_1^T A q_1) q_1 \]

\[ = q_1^T A q_1 (1 - q_1^T q_1) = 0. \]

Therefore, the vectors \( z \) and \( q_2 \) are orthogonal to \( q_1 \) by this choice of \( q_1 \). Moreover, since \( \beta_2 = ||z|| \), we have vector \( q_2 \) of unit length. By induction, it can be shown that the vector \( q_{i+1} \) is orthogonal to all the previous vectors \( q_i \) and that \( q_{i+1} \) is of unit length.

Thus, after \( k \) steps of the Lanczos algorithm we have the matrix equation,

\[ AQ_k = Q_k T_k + \beta_{k+1} q_k^* q_{k+1}^T. \]  

(II.3.5.20)
where $Q_k$ is the $(n \times k)$ rectangular matrix $[q_1, q_2, \ldots, q_k]$, $e_k$ is a vector with all its elements zero except the $k^{th}$ element being equal to unity and $T_k$ is the following tridiagonal matrix:

$$
T_k = \begin{bmatrix}
\alpha_1 & \beta_2 & & \\
\beta_2 & \alpha_2 & \beta_3 & \\
& \beta_3 & \alpha_3 & \beta_4 & \\
& & & \ddots & \ddots & \ddots \\
& & & & \beta_{k-1} & \alpha_{k-1} & \beta_k \\
& & & & & \beta_k & \alpha_k
\end{bmatrix}
$$

From (11.3.5.20) we can see that once $\beta_{k+1}$ ($k \leq n$) is zero, then the process is terminated and we have:

$$
Q_k^t A Q_k = T_k.
$$

The fact that in an $n$-dimensional space we can find at most $n$ mutually orthogonal vectors suggests that the above process should reach a value $\beta_i = 0$ by the $n^{th}$ step. On the other hand, the starting vector $q_1$ must be chosen in such a way that it is not orthogonal to any of the eigenvectors of $A$ if the process is not to stop prematurely. Thus, if we assume that the Lanczos algorithm produces an orthogonal matrix $Q_n$, see LEWIS [75], where

$$
Q_n^t A Q_n = T_n
$$

then all the eigenvalues of $A$ can be evaluated by computing an LR factorisation (say) of $T$, or using the Sturm sequence to evaluate any particular eigenvalue lying in a specified range.

Although the Lanczos algorithm seems an effective strategy for the determination of the eigenvalues theoretically, it is not such a simple and straightforward process in practice. Indeed, the column vectors of
the orthogonal matrix $Q_k$ rapidly lose their orthogonality as $k$ increases and the process of the reorthogonalisation of each new column to all its predecessors can be very costly. These difficulties and some practical algorithms together with useful recommendations on this method are fully described in PAIGE [76-79] and LEWIS [75].

Among the methods described in this section, the gradient method has a reputation of being a slow process whereas the convergency of the other methods is guaranteed (in theory) after a maximum of $n$ iterations. The amount of work involved in these methods is equivalent to the amount of work required for one matrix-vector multiplication, i.e. A times a vector $q$, at each step of the process although in practice two such multiplications are often employed, see LUENBERGER [72].

In Chapter V we shall study the application of the standard iterative methods for the solution of linear systems (e.g. Gauss-Seidel, S.O.R., etc.) in evaluating eigenvalues and eigenvectors of sparse symmetric matrices by the minimisation of the Rayleigh quotient.
CHAPTER III

SPARSE GAUSSIAN-ELIMINATION WITH PIVOTAL STRATEGIES

AND THE DETERMINATION OF EIGENVALUES
III.1  INTRODUCTION

In the ensuing work the methods we use to obtain the eigenvalues and/or eigenvectors of a matrix are based on sparse Gaussian elimination procedures and so we devote this chapter to a full description of sparse elimination techniques and in particular symmetric Gaussian elimination.

We first describe in some detail the theory of Gaussian-elimination even though it is a widely known process. However, a detailed description of the method and some satisfactory error bounds can be found in WILKINSON [9].

Gaussian elimination is generally used to solve a set of linear equations of the form

$$Ax = b$$  \hspace{1cm} (III.1.1)

by reducing $A$ to an upper triangular matrix. The elimination in the matrix $A$ takes place in $(n-1)$ stages, evaluating matrices $A^{(k)}$, $k=1,2,\ldots,n-1$ where in general after the $(k-1)^{th}$ stage of elimination, the matrix $A^{(k-1)}$ has the form,

$$A^{(k-1)} = \begin{bmatrix}
    a_{1,1} & a_{1,2} & a_{1,3} & \ldots & a_{1,n} \\
    0 & a_{2,2} & a_{2,3} & \ldots & a_{2,n} \\
    0 & a_{3,3} & a_{3,4} & \ldots & a_{3,n} \\
    \vdots & \vdots & \vdots & \ddots & \vdots \\
    0 & 0 & 0 & \ldots & a_{n,n}
\end{bmatrix}$$  \hspace{1cm} (III.1.2)

where the elements $a_{i,k}^{(k-1)}$, $i=k+1,k+2,\ldots,n$ are now eliminated by calculating

$$a_{i,j}^{(k)} = a_{i,j}^{(k-1)} - \frac{a_{i,k}^{(k-1)}a_{k,j}^{(k-1)}}{a_{k,k}^{(k-1)}}, \quad i,j=k+1,k+2,\ldots,n.$$  \hspace{1cm} (III.1.3)

It can be seen that after the $(k-1)$ stages of elimination any
one of the elements $a_{i,j}^{(k-1)}$, $i,j=k,k+1,...,n$ could be chosen as the new pivot. Pivotal strategies are employed in order to maintain accuracy and retain sparsity when $A$ is originally sparse. These strategies are fully discussed in later sections.

It usually happens that during the elimination process new non-zero elements (fill-ins) are created and therefore, the major purpose of this chapter is to study the effect of the elimination process on the sparsity of the matrices in question and to minimise the amount of fill-ins by using some types of pivotal strategies. The following example illustrates the important role of a suitable choice of pivoting in order to minimise the number of non-zero elements created during the process. Suppose that the matrix $A$ is of the form as follows:

$$A = \begin{bmatrix}
X & X & X & X & X & \cdots & X \\
X & X & X & X & X & \cdots & X \\
X & X & X & X & 0 & \cdots & 0 \\
X & X & X & X & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
X & X & X & X & X & \cdots & X \\
\end{bmatrix} \quad (III.1.4)$$

where $(X)$ denotes the position of a non-zero element.

If we apply the Gaussian elimination method (III.1.3) on $A$ without any re-ordering (rows or columns) we will have a full matrix by the end of the process. However, if we permute the matrix to have the form as follows:

$$B = P A P^T = \begin{bmatrix}
X & X & X & X & X & \cdots & X \\
X & X & X & X & X & \cdots & X \\
0 & X & X & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & X & \cdots & \cdots & \cdots & \cdots \\
\vdots & \vdots & \vdots & \ddots & \cdots & \cdots & \cdots \\
X & X & X & X & X & \cdots & X \\
\end{bmatrix} \quad (III.1.5)$$

...
where $P$ is a permutation matrix defined as,

$$
P = \begin{bmatrix}
0 & 1 \\
1 & 0 \\
1 & 0
\end{bmatrix} \quad \text{(III.1.6)}
$$

then there will be no fill-ins during the elimination process.

The above situation where we have an ordering beforehand, which when operated on the matrix minimises the number of new non-zero elements (fill-ins) occurs rarely in practice. However, an ordering which locally minimises the amount of fill-ins, as described in later sections, can be used.
III.2  **Symmetric Gaussian Elimination and Local Minimisation Algorithms**

In this section we describe the Crout form of factorisation for a symmetric positive definite matrix, $A = U^T DU$. (Isaccson & Keller [21], Rose [22], Sherman [25]).

The $U^T DU$ factorisation of the matrix $A$ is obtained in a recursive procedure where at the first step of the process, we write $A$ as:

$$
A = A^{(1)} = \begin{bmatrix} a_1 & r_1^t \\ r_1 & C_1 \end{bmatrix}
$$

where $a_1$ is a single element, $r_1$ is a $(n-1) \times 1$ vector and $C_1$ is a $(n-1) \times (n-1)$ matrix. The first element of the diagonal matrix $D$, i.e. $d_{1,1}$, and the first row of the upper triangular matrix $U$ is now computed as:

$$
u_{i,j} = [r_1^t / a_1]_j, \quad j=2,3,\ldots,n$$

and

$$d_{1,1} = a_1.$$ 

We then compute $A^{(2)}$ as:

$$
A^{(2)} = \begin{bmatrix} a_1 & r_1^t \\ r_1 & C_1 - \frac{r_1^t r_1}{a_1} \end{bmatrix}
$$

where the matrix $\frac{r_1^t r_1}{a_1}$ is subtracted element by element from $C_1$.

In general, at the $k^{th}$ step ($1 \leq k \leq n-1$) we have $A^{(k)}$ as:
where $A_k$ is a $(k-1) \times (k-1)$ matrix, $B_k$ is a $(n-k+1) \times (k-1)$ array, $a_k$ is a single element, $r_k$ is $(n-k) \times 1$ vector and $C_k$ is a $(n-k) \times (n-k)$ matrix. We now have $d_k = a_k$ and the $k^{th}$ row of the matrix $U$ is replaced by $r_k/a_k$.

Finally, we compute $A^{(k+1)}$ as:

```
A^{(k+1)} =
\begin{bmatrix}
A_k & B^t_k \\
B_k & r_k
\end{bmatrix}
```

For our further discussion, it is of great convenience to visualise the elimination process by representing the structure of the matrix in a graph since the process of vertex elimination on graphs models the process of performing Gaussian elimination on the matrix, c.f. PARTER [23], ROSE [22].

The structure of an $(n \times n)$ symmetric matrix $A$ is given by the graph

$G(A) = (X(A), E(A))$
where
\[ X(A) = \{x_1, x_2, \ldots, x_n\} \]
is the set of vertices and
\[ E(A) = \\{ \{x_i, x_j\} \mid i \neq j, \ a_{i,j} \neq 0 \} \]
is the set of edges. For example, if \( A \) is
\[
A = \begin{bmatrix}
  a_{1,1} & 0 & a_{1,3} & a_{1,4} & 0 & a_{1,6} \\
  0 & a_{2,2} & a_{2,3} & a_{2,4} & 0 & 0 \\
  a_{3,1} & a_{3,2} & a_{3,3} & 0 & a_{3,5} & a_{2,6} \\
  a_{4,1} & a_{4,2} & 0 & a_{4,4} & a_{4,5} & 0 \\
  0 & 0 & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} \\
  a_{6,1} & 0 & a_{6,3} & 0 & a_{6,5} & a_{6,6}
\end{bmatrix}
\]
then, the corresponding graph is shown below in Figure (III.2):

\begin{center}
\textbf{FIGURE (III.2)}
\end{center}

In the above graph, the vertex \( x_i \) corresponds to row \( i \) (or column \( i \)) and the set of edges i.e.
\[
E(A) = \\{ \{x_1, x_6\}, \{x_1, x_3\}, \{x_1, x_4\}, \{x_6, x_3\}, \{x_3, x_2\}, \{x_3, x_4\},
\{x_6, x_5\}, \{x_3, x_5\}, \{x_5, x_4\} \}
\]
corresponds to the non-zero elements of \( A \).

For any vertex \( x_i \in X(A) \), we define the set of vertices which are adjacent to \( x_i \) as,
Adj\( (x_i) \) = \{x_j \mid x_j \in X(A), x_j \text{ is adjacent to } x_i\}

and the degree of \( x_i \) as
\[ \text{Deg}(x_i) = |\text{Adj}(x_i)|. \]

The deficiency of a vertex \( x_i \), \( \text{def}(x_i) \), is the set of all pairs of distinct vertices \( \{x_j, x_k\} \) where \( x_j \) and \( x_k \) are not adjacent to each other and are both adjacent to \( x_i \). For example, in Figure (III.2) we have

\[ \text{Adj}(x_1) = \{x_3, x_4, x_6\} \quad \text{Deg}(x_1) = 3 \]
\[ \text{Adj}(x_3) = \{x_2, x_6\} \quad \text{Deg}(x_3) = 2 \]

and
\[ \text{Def}(x_3) = \{(x_1, x_5), (x_2, x_5), (x_1, x_2)\} \]
\[ \text{Def}(x_1) = \{(x_3, x_4), (x_4, x_6)\}. \]

If, in the Crout method as described above, the element \( a_{i,j} \) is re-ordered so that it is chosen as the pivot in the first elimination step, then the \( x_1 \)-elimination graph of \( G \) as graph \( G^{(2)} \) is formed by:

i) deleting vertex \( x_1 \) and all edges upon it,

ii) joining all pairs of vertices each of which belonged to \( \text{Adj}(x_1) \) in \( G^{(1)} \).

For the Graph \( G \) shown in Figure (III.2), if the vertex \( x_5 \) (corresponding to the element \( a_{5,5} \) of the matrix \( A \)) is re-ordered as \( x_1 \), then the graph \( G^{(2)} \) which is formed by eliminating the vertex \( x_1 \) according to the above algorithm is given below.

FIGURE (III.3)
It can be seen that graph $G^{(2)}$ is not a subgraph of $G$, c.f. HARY [23].

Indeed, the graph elimination process can be defined as a sequence of graphs $G^{(1)}, G^{(2)}, \ldots, G^{(n)}$ where,

$$G^{(k)} = \{x_k, E_k\}$$

with

$$x_k = \{x_i \mid x_i \in X \& i \neq k\}, \quad E_k = E_{k-1} \cup \text{Def}(x_k) = \frac{E_{k-1}}{x_k}$$

where

$$\frac{E_{k-1}}{x_k} = \{(x_{k'}, x_j) \in E \mid x_j \in \text{adj}(x_k)\}.$$ 

Comparing the two graphs, $G^{(1)}, G^{(2)}$, we can see that there are two edges i.e. $\{x_3, x_4\}, \{x_4, x_6\}$, which have been created after the first step of the elimination procedure and their set is the deficiency of vertex $x_5$ in $G^{(1)}$. In general, at the $k$th step, the number of such vertices (fill-ins) which are created as a result of eliminating $x_k$ from the graph $G^{(k)}$ is the Def$(x_k)$ in $G^{(k)}$. This fact is the basis of the first ordering we shall discuss in this section in order to minimise the storage, by creating as few non-zero elements as possible at each step of elimination.

This ordering is called the minimum deficiency algorithm, ROSE [22]. In this algorithm at the $k$th step of the elimination process, we search among the remaining unordered vertices and a vertex with the smallest deficiency is ordered as $x_k$ and the elimination proceeds using $x_k$ as the new pivot.

The second ordering which is described here, locally minimises the computational cost by reducing the number of arithmetic operations. The method is called the minimum degree algorithm, see [24]. It can be shown, SHERMAN [25], that the total number of multiplications or divisions i.e. $\theta_M$ and the total number of additions or subtractions, i.e. $\theta_A$ required for the $U^T \cdot D \cdot U$ factorisation of $A$ are:

$$\theta_M = \sum_{k=1}^{n-1} \frac{\text{Deg}(x_k) \cdot (\text{Deg}(x_k) + 3)}{2}$$
\[ \theta_A = \sum_{k=1}^{n-1} \frac{\text{Deg}(x_k)(\text{Deg}(x_k)+1)}{2} \]

and to minimise this computational cost, at the \( k \text{th} \) step of the elimination process, a vertex is reordered as \( x_k \) which has the minimum degree among the remaining vertices in \( G^{(k)} \).

In both algorithms described above, tie-breaking strategies must be applied when searching for a vertex of minimum deficiency or minimum degree, if there are several vertices which all have the same smallest deficiency or minimum degree.
In the previous section we discussed two pivotal strategies for sparse symmetric matrices which locally minimise either the storage requirement or computational cost of the factorisation process. In this section it is assumed that $A^k = (a_{i,j})_{i,j=1,2,\ldots,n}$ is a general matrix and that $(k-1)$ steps of the elimination process have been completed and the $k^{th}$ pivotal element is to be found.

We define the number of non-zero elements in row $i$ ($i=k,k+1,\ldots,n$) of $A$ with their column number greater than or equal to $k$ as $r(i,k)$, i.e.

$$r(i,k) = \{a_{i,j} \mid j \geq k \land a_{i,j} \neq 0, i=k,k+1,\ldots,n\}$$  \hspace{1cm} (III.3.1)

and the number of non-zero elements in column $j$ ($j=k,k+1,\ldots,n$) with their row number greater than or equal to $k$ as $c(j,k)$, i.e.

$$c(j,k) = \{a_{i,j} \mid i \geq k \land a_{i,j} \neq 0, j=k,k+1,\ldots,n\}.$$  \hspace{1cm} (III.3.2)

We also define the submatrix $A_k$ of $A$ which contains all the elements $a_{i,j}$ where $i,j \geq k$ as the active part of $A$ at stage $k$.

Perhaps the most widely used pivotal strategy is that proposed by Markowitz [26] whereby the non-zero element $a_{i,j}$ of the matrix $A_k$ which minimises the quantity:

$$(r(i,k)-1)*(c(j,k)-1)$$  \hspace{1cm} (III.3.3)

is chosen as the pivot at the $k^{th}$ stage. This method locally minimises the number of new non-zero elements (fill-ins) created at step $k$ of the elimination process since the quantity (III.3.3) is an upper bound on the fill-in. On the other hand, the choice of a non-zero element $a_{i,j}$ of the matrix $A_k$ which minimises the quantity:

$$r(i,k)(c(j,k)-1)+1$$  \hspace{1cm} (III.3.4)

locally reduces the number of arithmetic operations since quantity (III.3.4) represents the number of multiplications and divisions required at stage $k$ of the elimination process.

It can be shown, Duff & Reid [30], that the use of expression...
(III.3.3) gives virtually identical result to the use of expression (III.3.4).

ZLATER [27] has recently developed the new pivotal strategies known as the generalised Markowitz algorithm which depend on the following two parameters:

1. \( U \) - the stability factor
2. \( p(k) \) - the number of rows that will be investigated for a pivotal element at stage \( k \) of the elimination process.

and shows that pivotal strategies with small \( p(k) \) may be more profitable than the original Markowitz algorithm.

Since pivotal strategies of this type have been used for our numerical experiments in the next chapter, we shall describe these methods in more detail. However, a full description of the strategies can be found in ZLATER [27].

We define \( B_k \) as a set of the non-zero elements of \( A_k \) at stage \( k \) as:

\[
B_k = \{ a_{i,j}^{(k)} \mid a_{i,j} \in A_k, \quad \text{s.t.} \quad a_{i,j}^{(k)} > \max_{1 \leq \ell \leq n} \left| a_{i,j}^{(k)} \right|/U, \quad i \in I_k \}
\]

where

\[
U > 1, \quad I_k = \{ i_1, i_2, \ldots, i_{p(k)} \},
\]

for \( k \leq i \leq n \), \( m = 1, 2, \ldots, p(k) \)

with

\[
r(i_1, k) \leq r(i_2, k) \leq \ldots \leq r(i_{p(k)}, k),
\]

and if \( i \notin I_k \) and \( k \leq i \leq n \) then \( r(i_{p(k)}, k) \leq r(i, k) \).

The parameter \( U \) is called a stability factor and if \( U > 1 \) and \( A \) is a non-singular matrix then the set \( B_k \) whose elements are said to satisfy the stability condition will not be empty.

In the generalised Markowitz algorithm any member of the set \( B_k \) for which the quantity (III.3.3) is minimum can be chosen as the pivotal element at stage \( k \). We define the set of such elements as \( C_k \).

As we mentioned above, the choice of a pivotal element according to
the generalised Markowitz algorithm depends on two parameters: $U$ and $p(k)$ and when $U = \infty$ and $p(k) = n-k+1$ ($k=1,2,\ldots,n$) the algorithm is equivalent to the original Markowitz strategy. The suggested values for the two parameters $U$ and $p(k)$ in two sparse elimination programs are as follows:

i) MA28 (c.f. DUFF [28] and DUFF & REID [29]) - $4\varepsilon U \leq 10$ and $p(k) = n-k+1$

ii) The FORTRAN version of SSLEST (c.f. [31]) - $4\varepsilon U \leq 16$ and

$p(s) = \min(RPIV, n-k+1)$, where RPIV may be $1,2,\ldots,n$.

It can be shown (c.f. [37]) that if the above pivotal strategy is employed to determine the pivot at each stage of the elimination process then the following bound can be derived:

$$b \leq (1+U)^{n-1}a,$$  \hspace{1cm} (III.3.6)

where $a$ is the largest element in absolute value of the original matrix $A$, and $b$ is the largest element in absolute value created at any stage of the elimination. On the other hand, it can be proved (c.f. REID [35]) that if an approximation LU-factorisation of $A$ where $L$ and $U$ are lower and upper triangular matrices respectively is carried out so that:

$$LU = PAQ + E,$$  \hspace{1cm} (III.3.7)

then the elements of the perturbation matrix $E$, i.e. $e_{i,j}$ satisfy the inequality

$$|e_{i,j}| \leq 3.01 n \epsilon$$ \hspace{1cm} (III.3.8)

where $\epsilon$ is the machine accuracy. Therefore, to keep the elements $e_{i,j}$ small enough to obtain an accurate LU-factorisation of $A$, the value $b$ must be kept small. Obviously, the right-hand side of (III.3.6) may be reduced by the choice of a smaller value of the stability parameter $U$. However, decreasing the value of $U$ may cause the number of elements in $B_k$ to decrease too, and this may force one to choose an element of $B_k$ as the pivot for which the quantity (III.3.3) is not small enough to keep the amount of fill-ins down.

Another pivotal strategy which sometimes gives a possibility of
reducing the right-hand side of (III.3.6) is to choose the largest element of the set $C_k$ as a pivotal element. This is called the improved generalised Markowitz strategies. The reasons for using the term "improved" for this strategy and some examples of the application of the above two strategies can be found in [27].
III.4 NESTED DISSECTION ALGORITHMS

In this section we consider the ordering strategies for a finite element system of equations associated with a regular n*n mesh or grid. We start by giving an example which illustrates how the way we number or order the vertices of an n*n mesh, D_n, can affect the amount of computation and storage requirement for the Gaussian elimination of the corresponding matrix.

Suppose that the internal points of a 3*3 mesh, D_3, are ordered as follows:

```
  1  2  3
  4  5  6
  7  8  9
```

**FIGURE (III.4.1)**

then, the non-zero elements of the matrix derived from the 5 point finite difference analogue of the Laplace equation are marked by an 'X' in Figure (III.4.2).

```
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>.</td>
<td>X</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>.</td>
<td>.</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>.</td>
<td>.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>.</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>.</td>
<td>X</td>
<td>X</td>
<td>.</td>
<td>.</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>.</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>.</td>
<td>.</td>
<td>X</td>
<td>X</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>.</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>.</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**FIGURE (III.4.2)**
Ignoring the accidental creation of zero elements during the elimination process, it can easily be seen that the number of the non-zero elements created during the elimination process whose positions are marked by a (.) in Figure (III.4.2) is sixteen.

However, if we re-number the above region as follows:

\[
\begin{array}{ccc}
1 & 2 & 4 \\
3 & 5 & 7 \\
6 & 8 & 9 \\
\end{array}
\]

**FIGURE (III.4.3)**

then, Figure (III.4.4) shows the structure of the corresponding matrix and the occurrence of fill-ins. It can be seen that the number of non-zero elements which are created as a result of the elimination is now thirteen. For larger mesh problems, this difference would be very much greater.

\[
\begin{array}{cccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
\hline
1 & X & X & X & . & . & . & . & . \\
2 & X & X & X & X & . & . & . & . \\
3 & X & . & X & . & X & . & . & . \\
4 & X & . & X & . & . & X & . & . \\
5 & X & X & . & X & . & X & . & . \\
6 & X & . & . & X & . & X & . & . \\
7 & X & X & . & X & . & X & . & . \\
8 & . & X & X & . & X & . & X & . \\
9 & . & . & . & X & X & X & . & . \\
\end{array}
\]

**FIGURE (III.4.4)**
We now turn to near optimal orderings of the vertices of a mesh produced by GEORGE [33], e.g. for a nine-point model problem (see also WOO, ROBERT and GUSTAVYSON [32], BIRKHOFF and GEORGE [34]) and briefly describe the algorithm for an \( (n \times n) \) mesh where \( n = 2^t - 1 \) for some positive integer \( t \), c.f. SHERMAN [25]. For other values of \( n \), see [36] and [40].

The nested dissection algorithm essentially involves successive dissections of successive subregions where the basic step of the process consists of first choosing a separating cross which divides the submesh into identical \( m \times m \) quadrants, i.e.,

![Separating cross in \( D_7 \)](image)

then ordering the vertices on the separating cross is completed as follows.

If the vertices \( x_{k+1}, x_{k+2}, \ldots, x_n \) have already been determined then the \( 2m-1 \) vertices along the separating cross in any submesh \( D_m \) are ordered as \( x_{k-2m+2}, x_{k-2m+3}, \ldots, x_k \). See Figure (III.4.6). Note that the ordering within each separating cross is relatively unimportant.
The algorithms start by applying the basic step as described above to the mesh $D_n$ and divides it into four identical $(m^2 \times m^2)$ submeshes where $m = 2^{t-1} - 1$, see Figure (III.4.7). The process is now repeated by applying the basic step to each of the submeshes $D_m$ to obtain 16 square submeshes $D_m'$ where $m' = 2^{t-2} - 1$ and continues in this way until all the vertices on the mesh have been ordered. Figure (III.4.8) illustrates the nested dissection ordering on a $(7 \times 7)$ grid.

Numbering of vertices in separating cross in $D_7$

**FIGURE (III.4.6)**

**FIGURE (III.4.7)**
Note that in the above algorithm, the basic ordering step is applied to all the submeshes of equal size before operating on any smaller submeshes. However, ROSE and WHITTEN [40] have produced a version of the nested dissection algorithm whereby at the first step we divide the mesh $D_n$ into four identical $m \times m$ submeshes $D_m$ where $m = 2^{t-1} - 1$ by applying the basic step to $D_n$. Next, the basic step is applied to one of the generated $D_m$'s to obtain four smaller square submatrices $D_m'$ where $m' = 2^{t-2} - 1$. 
then one of the $D^i_n$ is further subdivided and the algorithm continues in this way recursively until all the vertices in $D^i_n$ have been ordered. This strategy is called the depth-first nested dissection whereas the first method is called the breadth-first nested dissection. The two algorithms differ only in the order in which they process the submeshes. However both have an equivalent set of separating crosses. Figure (III.4.9) illustrates the depth-first nested dissection ordering on a (7*7) grid.

![Diagram](image)

**FIGURE (III.4.9)**
It can be shown, GEORGE [33], that the number of non-zero elements of the final upper triangular matrix of the elimination process is $O(n^2 \lg n)$ for an $n \times n$ mesh, and the amount of arithmetic operations required is $O(n^3)$. In the natural ordering of the vertices these numbers are, $O(n^3)$ and $O(n^4)$ respectively.
III.5 DETERMINATION OF THE EIGENVALUES OF A MATRIX BY THE ELIMINATION METHOD

Now that sparse elimination techniques and the significant role of pivotal strategies have been fully explained, we outline the procedure for determining the eigenvalues of a matrix $A$ by a suitable modification to the Gaussian elimination process together with the use of polynomial solvers.

In this section we describe how the eigenvalues of a matrix can be obtained by the elimination method and therefore, the modifications required to convert a linear system solver program package to an eigenvalue system solver.

We first prove that the determinant of an upper triangular matrix $A$, is equal to the product of the elements along the leading diagonal, i.e.

$$\det(A) = \prod_{i=1}^{n} a_{i,i} \quad (III.5.1)$$

The proof is obtained easily by induction, where for $n=2$, using the expansion of $\det(A_2)$ by its second row as described in (II.1.7) we have:

$$\begin{vmatrix}
a_{1,1} & a_{1,2} \\
0 & a_{2,2}
\end{vmatrix} = a_{1,1}a_{2,2} \quad (III.5.2)$$

We now assume that for $n=m$ we have

$$\det(A_m) = \prod_{i=1}^{m} a_{i,i} \quad (III.5.3)$$

and prove that for $n=m+1$ the following relationship holds:

$$\det(A_{m+1}) = \prod_{i=1}^{m+1} a_{i,i} \quad (III.5.4)$$

The matrix $A_{m+1}$ can be written as:
Then using the expansion of \( \det(A_{m+1}) \) by its \((m+1)\)st row we have:

\[
\det(A_{m+1}) = a_{m+1,m+1} \cdot (-1)^{(m+1)} \cdot (m+1) \cdot \det(A_m)
\]

(III.5.6)

\[
= a_{m+1,m+1} \cdot (-1)^{2(m+1)} \cdot \det(A_m) = a_{m+1,m+1} \cdot \det(A_m)
\]

and substituting for \( \det(A_m) \) in the above expression its equivalent value in (III.5.3) we have:

\[
\det(A_{m+1}) = a_{m+1,m+1} \cdot \prod_{i=1}^{m} a_i \cdot i = \prod_{i=1}^{m+1} a_i \cdot i
\]

(III.5.7)

and the result (III.5.4) is obtained.

From the above discussion and the properties of determinants given in (II.1) we can conclude that once a matrix \( A \) has been factorised and the pivotal elements together with the total number of interchanges made during the factorisation process are known then

\[
\det(A) = (-1)^p \prod_{i=1}^{n} \text{pivot}(i)
\]

(III.5.8)

where \( p \) is the total number of interchanges and \( \text{pivot}(i) \), \( i=1,2,...,n \) denote the pivotal elements of \( A \).

We are now able to complete the determination of an eigenvalue \( \lambda \) of a matrix \( A \) by the elimination method with the help of polynomial solvers (i.e. Newton-Raphson iteration, Muller's method, etc.) The
problem is to find a value of $\lambda$ for which we have

$$F(\lambda) \equiv \det(A-\lambda I) = 0.$$  \hspace{1cm} (III.5.9)

To determine such a value we proceed as follows:

1. An initial guess to $\lambda$ (i.e. $\lambda_0$) is made and the determinant of $(A-\lambda_0 I)$ is evaluated by the elimination procedure and multiplying all the pivots together as described above.

2. If for this value of $\lambda$ the $\det(A-\lambda_0 I)$ is equal to zero then $\lambda_0$ is the eigenvalue required and the process is terminated.

3. Otherwise, an improved estimate to $\lambda$ (i.e. $\lambda_1$) is obtained from $\lambda_0$ and the determinantal value $F(\lambda_0)$ and its derivative $F'(\lambda_0)$, if required, together with previous evaluations, i.e. $\lambda_{-1}$, $F(\lambda_{-1})$, $F'(\lambda_{-1})$, $\lambda_{-2}$, $F(\lambda_{-2})$, $F'(\lambda_{-2})$, etc. by using one of the many well known polynomial solvers.

4. The process is repeated by replacing $\lambda_1$ for $\lambda_0$ in the above order until the difference between two successive values of $\lambda$ (i.e. $\lambda_1, \lambda_{i+1}$) is smaller than some predefined tolerance $\epsilon$.

Note that the choice of the polynomial solver for the above algorithm is important because of the arbitrary interchanges made during the elimination process. In the above two methods (i.e. the Secant and Muller's methods) the correct sign for the evaluated determinant after the completion of the elimination process can be obtained easily from the formula (III.5.8), however, during the row/column interchanging process it can be shown that the sign of successive minors of the matrix $A$ may be disturbed and consequently their use in a Sturm sequence procedure using the well known bisection algorithm (which is less costly than the above two methods) can lead to erroneous results. Further research into this topic is necessary before any firm conclusions can be given.
CHAPTER IV

NUMERICAL EXPERIMENTS ON ELIMINATION METHODS
IV.1 INTRODUCTION

In this chapter we apply some of the factorisation procedures previously discussed in Chapter III using pivotal strategies in order to:

1. preserve the sparsity of the original matrix (i.e. to minimise the number of non-zero elements created during the factorisation process;

and 2. to ensure numerically stable computations,

by using some of the routines of the software package Y12M designed by WASNIEWSKI and ZLATER [37]. The package was originally designed for the solution of large sparse systems of linear algebraic equations but was modified in order to make it suitable for the determination of eigensystems as described in Section (III.5). However, all the facilities provided in Y12M have been preserved in our programs and a full description can be found in [37] but a brief description is included in Appendix 1.

The programs were tested on a number of test cases and a description of the numerical experiments carried out is given below.

IV.2 NUMERICAL EXAMPLES

Example IV.2.1:

Consider the matrix $A_1$ given in (IV.2.2). The theoretical values of the eigenvalues of $A_1$ are known and are given by:

$$\lambda_i = \frac{1}{2} [1 - \cos(2i-1)\pi/25]^{-1}, \quad i=1,2,\ldots,12. \quad (IV.2.1)$$

The theoretical results are tabulated in Table (IV.2.1) together with experimental results obtained by using both the Muller's and Secant methods for determining the zeros of the $\det(A-\lambda I)$ implicitly. It can be seen that close agreements with the theoretical results was obtained.
\[ A_1 = \begin{bmatrix}
78 & -1001 & 5005 & -12870 & 19448 & -18564 & 11628 & -4845 & 1330 & -231 & 23 & -1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
\end{bmatrix} \]
Table (IV.2.1) illustrates the improved eigenvalues of $A_1$ obtained by the Rayleigh quotient formula (i.e., Chapter 2).

The Improved Eigenvalues of $A_1$ by Rayleigh Quotient

<table>
<thead>
<tr>
<th>$\lambda_i$</th>
<th>Muller's Method</th>
<th>Secant Method</th>
<th>Theoretical Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.253987789</td>
<td>0.253989125</td>
<td>0.253989756</td>
</tr>
<tr>
<td>2</td>
<td>0.266477243</td>
<td>0.266479964</td>
<td>0.266480952</td>
</tr>
<tr>
<td>3</td>
<td>0.289184909</td>
<td>0.289189052</td>
<td>0.289189726</td>
</tr>
<tr>
<td>4</td>
<td>0.325539038</td>
<td>0.325554303</td>
<td>0.325557500</td>
</tr>
<tr>
<td>5</td>
<td>0.381966011</td>
<td>0.381965713</td>
<td>0.381965994</td>
</tr>
<tr>
<td>6</td>
<td>0.470459597</td>
<td>0.470459597</td>
<td>0.470459610</td>
</tr>
<tr>
<td>7</td>
<td>0.615294736</td>
<td>0.615294736</td>
<td>0.615294635</td>
</tr>
<tr>
<td>8</td>
<td>0.870745330</td>
<td>0.870745329</td>
<td>0.870745301</td>
</tr>
<tr>
<td>9</td>
<td>1.379021186</td>
<td>1.379021186</td>
<td>1.379021406</td>
</tr>
<tr>
<td>10</td>
<td>2.618033988</td>
<td>2.618033988</td>
<td>2.618032455</td>
</tr>
<tr>
<td>11</td>
<td>7.120122174</td>
<td>7.120122174</td>
<td>7.120128631</td>
</tr>
<tr>
<td>12</td>
<td>63.409138948</td>
<td>63.409138948</td>
<td>63.409790039</td>
</tr>
</tbody>
</table>

Table (IV.2.2) The selected eigenvectors of $A_1$ are given below in Table (IV.2.3) and were obtained by the inverse iteration techniques described in Chapter II.
Selected Eigenvectors of Example IV.2.1. Corresponding to $\lambda_i$

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$\lambda_1$</th>
<th>$\lambda_2$</th>
<th>$\lambda_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0000000283</td>
<td>0.000001182</td>
<td>0.000025250</td>
</tr>
<tr>
<td>2</td>
<td>0.000001116</td>
<td>0.000004089</td>
<td>0.000066106</td>
</tr>
<tr>
<td>3</td>
<td>0.000004298</td>
<td>0.000014143</td>
<td>0.000173070</td>
</tr>
<tr>
<td>4</td>
<td>0.000017317</td>
<td>0.000048910</td>
<td>0.000453103</td>
</tr>
<tr>
<td>5</td>
<td>0.000068184</td>
<td>0.000169132</td>
<td>0.001186241</td>
</tr>
<tr>
<td>6</td>
<td>0.000268457</td>
<td>0.000584861</td>
<td>0.003105619</td>
</tr>
<tr>
<td>7</td>
<td>0.001056973</td>
<td>0.002022449</td>
<td>0.008130618</td>
</tr>
<tr>
<td>8</td>
<td>0.004161513</td>
<td>0.006993622</td>
<td>0.021286235</td>
</tr>
<tr>
<td>9</td>
<td>0.016384700</td>
<td>0.024183918</td>
<td>0.055728089</td>
</tr>
<tr>
<td>10</td>
<td>0.064509796</td>
<td>0.083627886</td>
<td>0.145898032</td>
</tr>
<tr>
<td>11</td>
<td>0.253987788</td>
<td>0.289184866</td>
<td>0.381966009</td>
</tr>
<tr>
<td>12</td>
<td>0.999999999</td>
<td>0.999999999</td>
<td>0.999999999</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$\lambda_6$</th>
<th>$\lambda_8$</th>
<th>$\lambda_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00024988</td>
<td>0.218173840</td>
<td>0.999999999</td>
</tr>
<tr>
<td>2</td>
<td>0.00053115</td>
<td>0.250559875</td>
<td>0.381966011</td>
</tr>
<tr>
<td>3</td>
<td>0.001129018</td>
<td>0.287753340</td>
<td>0.145898033</td>
</tr>
<tr>
<td>4</td>
<td>0.002399819</td>
<td>0.330467855</td>
<td>0.055728090</td>
</tr>
<tr>
<td>5</td>
<td>0.005101011</td>
<td>0.379522973</td>
<td>0.021286266</td>
</tr>
<tr>
<td>6</td>
<td>0.010842613</td>
<td>0.435859902</td>
<td>0.008130618</td>
</tr>
<tr>
<td>7</td>
<td>0.023046853</td>
<td>0.500559565</td>
<td>0.003105620</td>
</tr>
<tr>
<td>8</td>
<td>0.048987955</td>
<td>0.574863338</td>
<td>0.001186241</td>
</tr>
<tr>
<td>9</td>
<td>0.104127870</td>
<td>0.660196867</td>
<td>0.000453103</td>
</tr>
<tr>
<td>10</td>
<td>0.221332229</td>
<td>0.758196867</td>
<td>0.000173070</td>
</tr>
<tr>
<td>11</td>
<td>0.470459593</td>
<td>0.870745328</td>
<td>0.000066106</td>
</tr>
<tr>
<td>12</td>
<td>0.999999999</td>
<td>0.999999999</td>
<td>0.000025250</td>
</tr>
</tbody>
</table>

TABLE (IV.2.3)
We now consider the matrix $A_2$ given below in IV.2.3 which is sparse and unsymmetric of order 12.

$$A_2 = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-36 & 0 & 13 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 25 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix} \quad \text{(IV.2.3)}$$

The characteristic equation for the determinant of $(A_2 - \lambda I)$ can be easily determined in canonical form as:

$$(\lambda^2 - 1)^2 (\lambda^2 - 4)(\lambda^2 - 9)(\lambda^2 - 16)(\lambda^2 - 25),$$

from which the theoretical values of the eigenvalues can be established immediately as:

$$\pm 1, \pm 2, \pm 3, \pm 4, \pm 5.$$ 

The tabulated values of the eigenvalues and selected eigenvectors are given in Tables (IV.2.4) and (IV.2.5) respectively.
<table>
<thead>
<tr>
<th>$\lambda_i$</th>
<th>The Muller's Method</th>
<th>The Secant Method</th>
<th>Theoretical Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1.000004780</td>
<td>-1.000004780</td>
<td>-1</td>
</tr>
<tr>
<td>2</td>
<td>1.00001286</td>
<td>1.00001286</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-0.999998906</td>
<td>-0.999998905</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>0.99999361</td>
<td>0.99999361</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>-2.00000012</td>
<td>-2.000000000</td>
<td>-2</td>
</tr>
<tr>
<td>6</td>
<td>1.99999993</td>
<td>2.000000000</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>-3.000000000</td>
<td>-3.000000000</td>
<td>-3</td>
</tr>
<tr>
<td>8</td>
<td>3.000000000</td>
<td>3.000000000</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>-4.000000000</td>
<td>-4.000000000</td>
<td>-4</td>
</tr>
<tr>
<td>10</td>
<td>4.000000000</td>
<td>4.000000000</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>-5.000000000</td>
<td>-5.000000000</td>
<td>-5</td>
</tr>
<tr>
<td>12</td>
<td>5.000000000</td>
<td>5.000000000</td>
<td>5</td>
</tr>
</tbody>
</table>

**TABLE (IV.2.4)**
### Selected Eigenvectors of Example IV.2.2. Corresponding to $\lambda_1$

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$\lambda_1$</th>
<th>$\lambda_2$</th>
<th>$\lambda_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$-0.875932219 \times 10^{-10}$</td>
<td>$-0.199384290 \times 10^{-12}$</td>
<td>$-0.125000009$</td>
</tr>
<tr>
<td>2</td>
<td>$-0.380891149 \times 10^{-11}$</td>
<td>$0.498461184 \times 10^{-11}$</td>
<td>$0.249999999$</td>
</tr>
<tr>
<td>3</td>
<td>$-0.875936224 \times 10^{-10}$</td>
<td>$0.458584391 \times 10^{-11}$</td>
<td>$-0.500000012$</td>
</tr>
<tr>
<td>4</td>
<td>$-0.380851097 \times 10^{-11}$</td>
<td>$0.199388870 \times 10^{-12}$</td>
<td>$0.999999999$</td>
</tr>
<tr>
<td>5</td>
<td>$-0.999985659$</td>
<td>$0.999999999$</td>
<td>$-0.15566213 \times 10^{-7}$</td>
</tr>
<tr>
<td>6</td>
<td>$0.999990439$</td>
<td>$-0.999999906$</td>
<td>$-0.143044627 \times 10^{-7}$</td>
</tr>
<tr>
<td>7</td>
<td>$-0.999995219$</td>
<td>$0.999997812$</td>
<td>$-0.168287817 \times 10^{-8}$</td>
</tr>
<tr>
<td>8</td>
<td>$0.999999999$</td>
<td>$0.999996718$</td>
<td>$-0.269260475 \times 10^{-7}$</td>
</tr>
<tr>
<td>9</td>
<td>$0.380876979 \times 10^{-12}$</td>
<td>$-0.199380535 \times 10^{-13}$</td>
<td>$0.315539629 \times 10^{-8}$</td>
</tr>
<tr>
<td>10</td>
<td>$-0.974961178 \times 10^{-10}$</td>
<td>$0.510424220 \times 10^{-11}$</td>
<td>$-0.403890720 \times 10^{-7}$</td>
</tr>
<tr>
<td>11</td>
<td>$0.152357309 \times 10^{-12}$</td>
<td>$-0.797514336 \times 10^{-14}$</td>
<td>$0.167326157 \times 10^{-8}$</td>
</tr>
<tr>
<td>12</td>
<td>$-0.952110297 \times 10^{-10}$</td>
<td>$0.498461180 \times 10^{-11}$</td>
<td>$-0.360616713 \times 10^{-7}$</td>
</tr>
</tbody>
</table>

### $x_i$ Values

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$\lambda_8$</th>
<th>$\lambda_{10}$</th>
<th>$\lambda_{12}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$0.037037074$</td>
<td>$-0.200121126 \times 10^{-8}$</td>
<td>$-0.414890974 \times 10^{-9}$</td>
</tr>
<tr>
<td>2</td>
<td>$0.111111142$</td>
<td>$-0.211944475 \times 10^{-8}$</td>
<td>$-0.421345971 \times 10^{-9}$</td>
</tr>
<tr>
<td>3</td>
<td>$0.333333352$</td>
<td>$-0.259237872 \times 10^{-8}$</td>
<td>$-0.45362095 \times 10^{-9}$</td>
</tr>
<tr>
<td>4</td>
<td>$0.999999999$</td>
<td>$-0.448411456 \times 10^{-8}$</td>
<td>$-0.614995871 \times 10^{-9}$</td>
</tr>
<tr>
<td>5</td>
<td>$0.316785621 \times 10^{-7}$</td>
<td>$-0.158946679 \times 10^{-8}$</td>
<td>$-0.346716632 \times 10^{-9}$</td>
</tr>
<tr>
<td>6</td>
<td>$0.403224875 \times 10^{-7}$</td>
<td>$-0.194381697 \times 10^{-8}$</td>
<td>$-0.411096040 \times 10^{-9}$</td>
</tr>
<tr>
<td>7</td>
<td>$0.388976646 \times 10^{-7}$</td>
<td>$-0.188986743 \times 10^{-8}$</td>
<td>$-0.402371298 \times 10^{-9}$</td>
</tr>
<tr>
<td>8</td>
<td>$0.346231959 \times 10^{-7}$</td>
<td>$-0.167406942 \times 10^{-8}$</td>
<td>$-0.358747591 \times 10^{-9}$</td>
</tr>
<tr>
<td>9</td>
<td>$-0.403021374 \times 10^{-7}$</td>
<td>$0.249999999$</td>
<td>$-0.815074550 \times 10^{-9}$</td>
</tr>
<tr>
<td>10</td>
<td>$-0.18758128 \times 10^{-6}$</td>
<td>$0.999999999$</td>
<td>$-0.293886027 \times 10^{-8}$</td>
</tr>
<tr>
<td>11</td>
<td>$-0.186708804 \times 10^{-7}$</td>
<td>$0.28511494 \times 10^{-8}$</td>
<td>$0.199999999$</td>
</tr>
<tr>
<td>12</td>
<td>$-0.128234067 \times 10^{-6}$</td>
<td>$0.163483339 \times 10^{-7}$</td>
<td>$0.999999999$</td>
</tr>
</tbody>
</table>

**TABLE (IV.2.5)**
Example IV.2.3:

By considering the problem defined in Section I.2 the determination of the vibrational modes of a membrane can be shown to be given by the eigenvalues of the following generalised problem:

\[ \text{det}(k - \lambda M) = 0 \]

where

\[
\begin{bmatrix}
-8 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & -8 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & -8 & 1 & 1 & 1 & 0 \\
1 & 1 & 1 & -8 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & -8 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & -8 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\( k = \frac{1}{3h} \)

and

\[
\begin{bmatrix}
16 & 4 & 4 & 1 & 0 & 0 & 0 & 0 \\
4 & 16 & 1 & 4 & 0 & 0 & 0 & 0 \\
4 & 1 & 16 & 4 & 4 & 1 & 0 & 0 \\
1 & 4 & 4 & 16 & 1 & 4 & 1 & 0 \\
0 & 0 & 4 & 1 & 16 & 4 & 0 & 0 \\
0 & 0 & 1 & 4 & 4 & 16 & 4 & 0 \\
0 & 0 & 0 & 1 & 0 & 4 & 16 & 4 \\
0 & 0 & 0 & 0 & 0 & 0 & 4 & 16 \\
\end{bmatrix}
\]

\( M = \frac{h}{36} \)

for the mesh size \( h=1 \).

The results obtained are given in the following tables:

<table>
<thead>
<tr>
<th>( \lambda_i )</th>
<th>The Muller's Method</th>
<th>The Secant Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.73189266</td>
<td>1.73189259</td>
</tr>
<tr>
<td>2</td>
<td>3.64822061</td>
<td>3.64822046</td>
</tr>
<tr>
<td>3</td>
<td>4.59570785</td>
<td>4.59570767</td>
</tr>
<tr>
<td>4</td>
<td>6.47471288</td>
<td>6.47471283</td>
</tr>
<tr>
<td>5</td>
<td>8.22329185</td>
<td>8.22329152</td>
</tr>
<tr>
<td>6</td>
<td>9.11485004</td>
<td>9.11484967</td>
</tr>
<tr>
<td>7</td>
<td>11.48707900</td>
<td>11.48707855</td>
</tr>
<tr>
<td>8</td>
<td>14.22882645</td>
<td>14.22882394</td>
</tr>
</tbody>
</table>

\[ \text{(IV.2.4)} \]

\[ \text{(IV.2.5)} \]

\[ \text{TABLE (IV.2.6)} \]
Some Eigenvectors of Example IV.2.3 Corresponding to $\lambda_1$

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$\lambda_1$</th>
<th>$\lambda_4$</th>
<th>$\lambda_6$</th>
<th>$\lambda_8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.63378300</td>
<td>0.47095520</td>
<td>-0.45165691</td>
<td>0.66522785</td>
</tr>
<tr>
<td>2</td>
<td>0.63838331</td>
<td>-0.54604402</td>
<td>-0.94629015</td>
<td>-0.67761756</td>
</tr>
<tr>
<td>3</td>
<td>0.92277686</td>
<td>0.82360475</td>
<td>0.97506985</td>
<td>-0.97388273</td>
</tr>
<tr>
<td>4</td>
<td>0.99999999</td>
<td>-0.76300756</td>
<td>0.99999999</td>
<td>0.99999999</td>
</tr>
<tr>
<td>5</td>
<td>0.67123261</td>
<td>0.67975475</td>
<td>-0.9467246</td>
<td>0.83350310</td>
</tr>
<tr>
<td>6</td>
<td>0.77349961</td>
<td>-0.58789039</td>
<td>-0.43411088</td>
<td>-0.99910329</td>
</tr>
<tr>
<td>7</td>
<td>0.45004317</td>
<td>-0.20041293</td>
<td>-0.02873026</td>
<td>0.44591567</td>
</tr>
<tr>
<td>8</td>
<td>0.12473655</td>
<td>0.99999999</td>
<td>0.02793575</td>
<td>-0.23340575</td>
</tr>
</tbody>
</table>

**TABLE (IV.2.7)**

Example IV.2.4:

The 5-point difference approximation to the two-dimensional Laplace partial differential equation yields a block tridiagonal sparse matrix of the form:

\[
A_4 = \begin{bmatrix}
B & -I & & & \\
-I & B & -I & & \\
& -I & B & -I & \\
& & -I & B & -I \\
& & & -I & B
\end{bmatrix}
\]

where

\[
B = \begin{bmatrix}
4 & -1 & & & \\
-1 & 4 & -1 & & \\
& -1 & 4 & -1 & \\
& & -1 & 4 & -1 \\
& & & -1 & 4
\end{bmatrix}
\]

with $b=10$ and $n=4$. 

\[
(n \times n)
\]

\[
(b \times b)
\]
A selection of the results obtained are tabulated below in Tables (IV.2.8) and (IV.2.9). The theoretical values of the eigenvalues of this matrix are known and are given by:

\[ \lambda_{i,j} = 4 \left( \sin^2 \frac{\pi i}{2(nb+1)} \right) + \sin^2 \frac{\pi j}{2(b+1)} \quad i=1,2,...,n \quad \text{and} \quad j=1,2,...,b \]

<table>
<thead>
<tr>
<th>( \lambda_i )</th>
<th>Muller's Method</th>
<th>The Secant Method</th>
<th>Theoretical Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.462980057</td>
<td>0.462980064</td>
<td>0.462980181</td>
</tr>
<tr>
<td>2</td>
<td>0.699458942</td>
<td>0.699458945</td>
<td>0.699459075</td>
</tr>
<tr>
<td>3</td>
<td>1.072244537</td>
<td>1.072244543</td>
<td>1.072244644</td>
</tr>
<tr>
<td>4</td>
<td>1.551138069</td>
<td>1.551135985</td>
<td>1.551136136</td>
</tr>
<tr>
<td>5</td>
<td>2.097336334</td>
<td>2.097336334</td>
<td>2.097336769</td>
</tr>
<tr>
<td>18</td>
<td>4.691687629</td>
<td>4.691687479</td>
<td>4.691688061</td>
</tr>
<tr>
<td>19</td>
<td>5.064473076</td>
<td>5.064473076</td>
<td>5.064473629</td>
</tr>
<tr>
<td>20</td>
<td>5.300951958</td>
<td>5.300951958</td>
<td>5.300951958</td>
</tr>
<tr>
<td>21</td>
<td>2.699048037</td>
<td>2.699048041</td>
<td>2.699048519</td>
</tr>
<tr>
<td>22</td>
<td>2.935526913</td>
<td>2.935526923</td>
<td>2.935527325</td>
</tr>
<tr>
<td>36</td>
<td>5.902663745</td>
<td>5.902663665</td>
<td>5.902664459</td>
</tr>
<tr>
<td>37</td>
<td>6.448864402</td>
<td>6.448864014</td>
<td>6.448864459</td>
</tr>
<tr>
<td>38</td>
<td>6.927755457</td>
<td>6.927755456</td>
<td>6.927756309</td>
</tr>
<tr>
<td>39</td>
<td>7.300541078</td>
<td>7.300541054</td>
<td>7.300542354</td>
</tr>
<tr>
<td>40</td>
<td>7.537020238</td>
<td>7.537019935</td>
<td>7.537020683</td>
</tr>
</tbody>
</table>

**TABLE (IV.2.8)**
Some Eigenvectors of Example IV.2.3 Corresponding to $\lambda_i$

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$\lambda_5$</th>
<th>$\lambda_{20}$</th>
<th>$\lambda_{37}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.618033942</td>
<td>-0.284629759</td>
<td>0.567964621</td>
</tr>
<tr>
<td>2</td>
<td>0.175910778</td>
<td>0.546200333</td>
<td>-0.471889057</td>
</tr>
<tr>
<td>3</td>
<td>-0.567964557</td>
<td>-0.763521049</td>
<td>-0.175890567</td>
</tr>
<tr>
<td>4</td>
<td>-0.337570357</td>
<td>0.918985901</td>
<td>0.618007271</td>
</tr>
<tr>
<td>5</td>
<td>0.471882022</td>
<td>-0.99999994</td>
<td>-0.337548455</td>
</tr>
<tr>
<td>6</td>
<td>0.471882024</td>
<td>0.999999995</td>
<td>-0.337577948</td>
</tr>
<tr>
<td>7</td>
<td>-0.337570370</td>
<td>-0.918985872</td>
<td>0.618031845</td>
</tr>
<tr>
<td>8</td>
<td>-0.567964563</td>
<td>0.763521034</td>
<td>-0.175907426</td>
</tr>
<tr>
<td>9</td>
<td>0.175910764</td>
<td>-0.546200285</td>
<td>-0.471874074</td>
</tr>
<tr>
<td>10</td>
<td>0.618039332</td>
<td>0.284629623</td>
<td>0.567955447</td>
</tr>
<tr>
<td>11</td>
<td>0.999999957</td>
<td>-0.175910711</td>
<td>-0.918977998</td>
</tr>
<tr>
<td>12</td>
<td>0.284629648</td>
<td>0.337570346</td>
<td>0.763519718</td>
</tr>
<tr>
<td>13</td>
<td>-0.918985936</td>
<td>-0.471882051</td>
<td>0.284614889</td>
</tr>
<tr>
<td>14</td>
<td>-0.546200331</td>
<td>0.567964442</td>
<td>-0.999975143</td>
</tr>
<tr>
<td>15</td>
<td>0.763521121</td>
<td>-0.618033864</td>
<td>0.546181953</td>
</tr>
<tr>
<td>16</td>
<td>0.763521123</td>
<td>0.618033970</td>
<td>0.546201439</td>
</tr>
<tr>
<td>17</td>
<td>-0.546200329</td>
<td>-0.567964479</td>
<td>-0.999989499</td>
</tr>
<tr>
<td>18</td>
<td>-0.918985911</td>
<td>0.471881930</td>
<td>0.284625092</td>
</tr>
<tr>
<td>19</td>
<td>0.284629656</td>
<td>-0.337570362</td>
<td>0.763508745</td>
</tr>
<tr>
<td>20</td>
<td>0.999999952</td>
<td>0.175910799</td>
<td>-0.918970555</td>
</tr>
<tr>
<td>21</td>
<td>0.999999999</td>
<td>0.175910760</td>
<td>0.918969209</td>
</tr>
<tr>
<td>22</td>
<td>0.284629700</td>
<td>-0.337570406</td>
<td>-0.763501310</td>
</tr>
<tr>
<td>23</td>
<td>-0.918985914</td>
<td>0.471881971</td>
<td>-0.284635753</td>
</tr>
<tr>
<td>24</td>
<td>-0.546200341</td>
<td>-0.567964459</td>
<td>0.999999999</td>
</tr>
<tr>
<td>25</td>
<td>0.763521077</td>
<td>0.618033872</td>
<td>-0.546203385</td>
</tr>
<tr>
<td>26</td>
<td>0.763521093</td>
<td>-0.618034073</td>
<td>-0.546186490</td>
</tr>
<tr>
<td>27</td>
<td>-0.546200336</td>
<td>0.567964594</td>
<td>0.999984393</td>
</tr>
<tr>
<td>28</td>
<td>-0.918985985</td>
<td>-0.471881985</td>
<td>-0.284624989</td>
</tr>
<tr>
<td>29</td>
<td>0.284629692</td>
<td>0.337570357</td>
<td>-0.763508007</td>
</tr>
<tr>
<td>30</td>
<td>0.999999982</td>
<td>-0.175910791</td>
<td>0.918981019</td>
</tr>
<tr>
<td>31</td>
<td>0.618034001</td>
<td>0.284629621</td>
<td>-0.567948741</td>
</tr>
<tr>
<td>32</td>
<td>0.175910849</td>
<td>-0.546200244</td>
<td>0.471861023</td>
</tr>
<tr>
<td>33</td>
<td>0.567964516</td>
<td>0.763521055</td>
<td>0.175928056</td>
</tr>
<tr>
<td>34</td>
<td>-0.337570391</td>
<td>-0.918985986</td>
<td>-0.618045919</td>
</tr>
<tr>
<td>35</td>
<td>0.471881947</td>
<td>0.999999999</td>
<td>0.337583351</td>
</tr>
<tr>
<td>36</td>
<td>0.471881949</td>
<td>-0.999999890</td>
<td>0.337555663</td>
</tr>
<tr>
<td>37</td>
<td>-0.337570390</td>
<td>0.918985866</td>
<td>-0.618023723</td>
</tr>
<tr>
<td>38</td>
<td>-0.567964518</td>
<td>-0.763521061</td>
<td>0.175908196</td>
</tr>
<tr>
<td>39</td>
<td>0.175910846</td>
<td>0.546200317</td>
<td>0.471874505</td>
</tr>
<tr>
<td>40</td>
<td>0.618033990</td>
<td>-0.284629685</td>
<td>-0.567954985</td>
</tr>
</tbody>
</table>

**TABLE IV.2.9**
Example IV.2.5:

An ill-conditioned example taken from [45] is defined to be:

\[ A_5 = (a_{i,j}), \quad i,j=1,2,\ldots,30, \]

where the elements are given by

\[
a_{i,j} = \begin{cases} 20 & \text{for } i=j , \\ 15 & \text{for } i-j = 1 , \\ 6 & \text{for } i-j = 2 , \\ 1 & \text{for } i-j = 3 , \\ 0 & \text{for } i-j \geq 3 , \\ \end{cases}
\]

except

\[
a_{1,1} = a_{1,2} = a_{2,1} = a_{30,30} = a_{29,30} = a_{30,29} = 14
\]

as shown below in (IV.2.8).

\[
\begin{bmatrix}
14 & 14 & 6 & 1 \\
14 & 20 & 15 & 6 & 1 \\
6 & 15 & 20 & 15 & 6 & 1 \\
1 & 6 & 15 & 20 & 15 & 6 & 1 \\
& & & & & & \vdots \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
1 & 6 & 15 & 20 & 15 & 6 & 1 \\
1 & 6 & 15 & 20 & 15 & 6 \\
1 & 6 & 15 & 20 & 14 \\
1 & 6 & 14 & 14
\end{bmatrix}_{(30 \times 30)}
\]

A selection of the results and their agreement with the theoretical values is shown in Tables (IV.2.10) and (IV.2.11).
<table>
<thead>
<tr>
<th>$\lambda_i$</th>
<th>The Muller's Method</th>
<th>The Secant Method</th>
<th>Theoretical Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>63.50871751</td>
<td>63.50871751</td>
<td>63.50875377</td>
</tr>
<tr>
<td>2</td>
<td>62.05491885</td>
<td>62.05491885</td>
<td>62.05493450</td>
</tr>
<tr>
<td>3</td>
<td>59.69755095</td>
<td>59.69755124</td>
<td>59.69756126</td>
</tr>
<tr>
<td>7</td>
<td>43.52194899</td>
<td>43.52194899</td>
<td>43.52196216</td>
</tr>
<tr>
<td>8</td>
<td>38.54370130</td>
<td>38.54370130</td>
<td>38.54372501</td>
</tr>
<tr>
<td>9</td>
<td>33.51743346</td>
<td>33.51743347</td>
<td>33.51744174</td>
</tr>
<tr>
<td>10</td>
<td>28.59445184</td>
<td>28.59445184</td>
<td>28.59445333</td>
</tr>
<tr>
<td>11</td>
<td>23.90749287</td>
<td>23.90749287</td>
<td>23.90750646</td>
</tr>
<tr>
<td>12</td>
<td>19.56536688</td>
<td>19.56536691</td>
<td>19.56536889</td>
</tr>
<tr>
<td>18</td>
<td>3.36619847</td>
<td>3.36619848</td>
<td>3.36619824</td>
</tr>
<tr>
<td>19</td>
<td>2.22443817</td>
<td>2.22443817</td>
<td>2.22443893</td>
</tr>
<tr>
<td>20</td>
<td>1.40196354</td>
<td>1.40196354</td>
<td>1.40196129</td>
</tr>
<tr>
<td>21</td>
<td>0.83608852</td>
<td>0.83608851</td>
<td>0.83608768</td>
</tr>
<tr>
<td>22</td>
<td>0.46640576</td>
<td>0.46690575</td>
<td>0.46690560</td>
</tr>
<tr>
<td>23</td>
<td>0.24071864</td>
<td>0.24071864</td>
<td>0.24071931</td>
</tr>
<tr>
<td>28</td>
<td>0.00077160</td>
<td>0.00077163</td>
<td>0.00077163</td>
</tr>
<tr>
<td>29</td>
<td>0.00008827</td>
<td>0.00006861</td>
<td>0.00006861</td>
</tr>
<tr>
<td>30</td>
<td>0.00000061</td>
<td>0.00000107</td>
<td>0.00000108</td>
</tr>
</tbody>
</table>

**Table (IV.2.10)**
### Selected Eigenvectors of Example IV.2.5 Corresponding to $\lambda_i$

<table>
<thead>
<tr>
<th>$x_i$</th>
<th>$\lambda_1$</th>
<th>$\lambda_{12}$</th>
<th>$\lambda_{30}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.101298337</td>
<td>0.938943478</td>
<td>-0.101091784</td>
</tr>
<tr>
<td>2</td>
<td>0.201557217</td>
<td>0.652202472</td>
<td>0.201153121</td>
</tr>
<tr>
<td>3</td>
<td>0.299747847</td>
<td>-0.485920620</td>
<td>-0.299164397</td>
</tr>
<tr>
<td>4</td>
<td>0.394862658</td>
<td>-0.989725238</td>
<td>0.394125159</td>
</tr>
<tr>
<td>5</td>
<td>0.485925644</td>
<td>-0.201556940</td>
<td>-0.485067076</td>
</tr>
<tr>
<td>6</td>
<td>0.572002376</td>
<td>0.849732000</td>
<td>0.571059574</td>
</tr>
<tr>
<td>7</td>
<td>0.652209589</td>
<td>0.791791990</td>
<td>-0.651224015</td>
</tr>
<tr>
<td>8</td>
<td>0.725724249</td>
<td>-0.299747534</td>
<td>0.724737064</td>
</tr>
<tr>
<td>9</td>
<td>0.791791996</td>
<td>-0.999996865</td>
<td>-0.790845016</td>
</tr>
<tr>
<td>10</td>
<td>0.849734886</td>
<td>0.394860383</td>
<td>0.848661444</td>
</tr>
<tr>
<td>11</td>
<td>0.898958346</td>
<td>0.725722382</td>
<td>-0.898199465</td>
</tr>
<tr>
<td>12</td>
<td>0.938957277</td>
<td>0.898958841</td>
<td>0.938334107</td>
</tr>
<tr>
<td>13</td>
<td>0.969321235</td>
<td>-0.101298276</td>
<td>-0.968852032</td>
</tr>
<tr>
<td>14</td>
<td>0.989738646</td>
<td>0.969317411</td>
<td>0.989430385</td>
</tr>
<tr>
<td>15</td>
<td>0.999999999</td>
<td>-0.572000600</td>
<td>-0.999851865</td>
</tr>
<tr>
<td>16</td>
<td>0.999999999</td>
<td>0.572003932</td>
<td>0.999999999</td>
</tr>
<tr>
<td>17</td>
<td>0.989738646</td>
<td>0.969321631</td>
<td>-0.989868975</td>
</tr>
<tr>
<td>18</td>
<td>0.969321235</td>
<td>0.101297492</td>
<td>0.969554948</td>
</tr>
<tr>
<td>19</td>
<td>0.938957277</td>
<td>-0.898955306</td>
<td>-0.939269918</td>
</tr>
<tr>
<td>20</td>
<td>0.898958346</td>
<td>-0.725723493</td>
<td>0.899320878</td>
</tr>
<tr>
<td>21</td>
<td>0.849734886</td>
<td>0.394864149</td>
<td>-0.850123277</td>
</tr>
<tr>
<td>22</td>
<td>0.791791996</td>
<td>0.999999999</td>
<td>0.792183705</td>
</tr>
<tr>
<td>23</td>
<td>0.725724249</td>
<td>0.299746858</td>
<td>-0.726102306</td>
</tr>
<tr>
<td>24</td>
<td>0.652209589</td>
<td>-0.791789744</td>
<td>0.652559250</td>
</tr>
<tr>
<td>25</td>
<td>0.572002376</td>
<td>-0.849734973</td>
<td>-0.572314141</td>
</tr>
<tr>
<td>26</td>
<td>0.485925644</td>
<td>0.201557103</td>
<td>0.486191668</td>
</tr>
<tr>
<td>27</td>
<td>0.394862658</td>
<td>0.989749003</td>
<td>-0.395079023</td>
</tr>
<tr>
<td>28</td>
<td>0.299747847</td>
<td>0.485929029</td>
<td>0.299911304</td>
</tr>
<tr>
<td>29</td>
<td>0.201557217</td>
<td>-0.652214248</td>
<td>-0.201666828</td>
</tr>
<tr>
<td>30</td>
<td>0.101298337</td>
<td>-0.938968136</td>
<td>0.101353178</td>
</tr>
</tbody>
</table>

**TABLE (IV.2.11)**
IV.3 CONCLUSION

The main objectives of this work have been to determine the eigenvalues and/or eigenvectors of large sparse matrices by the Gaussian elimination method and at the same time to use different forms of pivotal strategies in order to minimise the number of non-zero elements created during the elimination process. This minimisation was successfully achieved in the numerical experiments where for many of the test matrices we used, e.g., $A_1, A_2, A_3$ and $A_5$, there were no fill-ins at all during the elimination process whilst for the matrix $A_4$ we created only 15 new non-zero elements.

It is well known that very accurate computation in any Gaussian elimination process can only be guaranteed if pivotal strategies are used which result in all the multipliers being $\leq 1$. However, in sparse matrix elimination we seek to obtain a minimum number of fill-ins as well as accuracy in our computations, so compromises have to be made in order to accomplish both objectives at the same time and this may result in the selection of a multiplier $>1$ and consequently obtaining slightly less accurate results.

To illustrate the above point we performed the Gaussian elimination method on the matrix $A_1$ given by (IV.2.2) and interchanges were made so that all the multipliers were $\leq 1$. As a result, 55 new non-zero elements were obtained which filled-up the upper triangular part of $A_1$ at the completion of the process. The eigenvalues obtained from this experiment were found to be more accurate than the previous results obtained earlier and these are tabulated together with the theoretical solutions in the following Table.
<table>
<thead>
<tr>
<th>$\lambda_i$</th>
<th>Eigenvalues of $A_1$ obtained from the above experiments with 55 new non-zero elements</th>
<th>Previous evaluation of eigenvalues of $A_1$ as given in Table (IV.2.1) with no fill-ins</th>
<th>Theoretical solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.253989777</td>
<td>0.253987789</td>
<td>0.253989756</td>
</tr>
<tr>
<td>2</td>
<td>0.266480957</td>
<td>0.266477243</td>
<td>0.266480952</td>
</tr>
<tr>
<td>3</td>
<td>0.289189747</td>
<td>0.289184909</td>
<td>0.289189726</td>
</tr>
<tr>
<td>4</td>
<td>0.325557544</td>
<td>0.325539038</td>
<td>0.325557500</td>
</tr>
<tr>
<td>5</td>
<td>0.381966011</td>
<td>0.381966011</td>
<td>0.381965994</td>
</tr>
<tr>
<td>6</td>
<td>0.470459617</td>
<td>0.470459597</td>
<td>0.470459610</td>
</tr>
<tr>
<td>7</td>
<td>0.615294696</td>
<td>0.615294736</td>
<td>0.615294635</td>
</tr>
<tr>
<td>8</td>
<td>0.870745329</td>
<td>0.870745330</td>
<td>0.870745301</td>
</tr>
<tr>
<td>9</td>
<td>1.379021186</td>
<td>1.379021186</td>
<td>1.379021406</td>
</tr>
<tr>
<td>10</td>
<td>2.618033988</td>
<td>2.618033988</td>
<td>2.618032455</td>
</tr>
<tr>
<td>11</td>
<td>7.120122174</td>
<td>7.120122174</td>
<td>7.120128631</td>
</tr>
<tr>
<td>12</td>
<td>63.409138948</td>
<td>63.409138948</td>
<td>63.409790039</td>
</tr>
</tbody>
</table>

**TABLE (IV.3.1)**

*Note: The evaluation of the roots in order of increasing absolute magnitude will result in more accurate determination of the smaller roots, see [Wilkinson](#), pp. 464-465.*
CHAPTER V

ITERATIVE METHODS FOR THE EVALUATION OF THE

EIGENVALUES AND EIGENVECTORS OF SYMMETRIC MATRICES
V.1 INTRODUCTION

In the previous chapters, the determination of the eigenvalues and eigenvectors of large sparse matrices by elimination methods was studied and although it was shown that the problem of fill-ins and hence the storage of sparse matrices can be partially solved by establishing suitable pivotal strategies, the amount of computational work involved in locating such pivots at each step of the elimination process is very costly. Therefore, the elimination methods are only suitable if all or most of the eigenvalues are required or if the order of the matrix is moderate in size.

In this chapter, we consider the iterative methods for obtaining the eigen-solution of sparse matrices and propose a new preconditioning method whereby the largest/smallest eigenvalue and the corresponding eigenvector of a large sparse symmetric matrix is determined by the maximisation/minimisation of the Rayleigh quotient of the given matrix A defined in (11.3.5.1). Numerical experiments were carried out on the S.O.R. and the new method as a basis for comparison and these are given in the next chapter. It can be seen from the results obtained, that for most cases the proposed preconditioned method has a reduced computational cost over that of the S.O.R. method.

However, because the main theme of the iterative methods is concerned with the Rayleigh quotient of a matrix, we devote the next section to describing some important principles which are derived using the Rayleigh quotient \( u(x) \) defined as in (11.3.5.1). See also Bathe [59].
V.2 SOME PROPERTIES OF THE RAYLEIGH QUOTIENT

The first observation is that if the matrix A is symmetric then for any vector x we have:

\[ \lambda_{\text{min}} \leq u(x) \leq \lambda_{\text{max}} \]  

(V.2.1)

and it follows that we have for any vector x, if A is positive definite or negative definite, then \( u(x) > 0 \) or \( u(x) < 0 \) respectively, and for A being an indefinite matrix, \( u(x) \) can be negative, zero or positive.

For the proof of (V.2.1) we can use the fact that the n eigenvectors of A (i.e. \( x_1, x_2, \ldots, x_n \)) are linearly independent and that any arbitrary vector x can be expressed as,

\[ x = \sum_{i=1}^{n} a_i x_i \]  

(V.2.2)

where \( a_i \) are real numbers.

Assuming that the eigenvalues of A are distinct, then there exists a set of corresponding eigenvectors which are orthogonal and hence:

\[ x_i^t x_j = \delta_{i,j}, \quad i,j=1,2,\ldots,n \]  

(V.2.3)

where \( \delta_{i,j} \) is kronecker delta, i.e. \( \delta_{i,j} = 1 \) when \( i=j \), and \( \delta_{i,j} = 0 \) when \( i \neq j \).

Therefore, substituting for x into (II.5.5.1) and using the statement of the eigenvalue problem:

\[ A x_i = \lambda x_i \]  

(V.2.4)

we obtain:

\[ u(x) = \frac{\lambda_1 a_1^2 + \lambda_2 a_2^2 + \ldots + \lambda_n a_n^2}{a_1^2 + a_2^2 + \ldots + a_n^2} \]  

(V.2.5)

Hence, if \( \lambda_i \neq 0 \) we have:

\[ u(x) = \lambda_1 \frac{\frac{a_1^2}{a_1^2} + \frac{\lambda_2}{a_1 a_2} \frac{a_2^2}{a_2^2} + \ldots + \frac{\lambda_n}{a_1 a_n} \frac{a_n^2}{a_n^2}}{\frac{a_1^2}{a_1^2} + \frac{a_2^2}{a_2^2} + \ldots + \frac{a_n^2}{a_n^2}} \]  

(V.2.6)
and if \( \lambda_n \neq 0 \), then we have,

\[
u(x) = \frac{2\sum_{i=1}^{n} \left( \lambda_i^n / \lambda_n \right) \alpha_i^2}{\alpha_1^2 + \alpha_2^2 + \cdots + \alpha_n^2} = \lambda_n + O(\varepsilon^2)
\]  

(V.2.7)

Assuming that the eigenvalues are ordered, i.e. \( \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \), then the above two relations (i.e. (V.2.6) and (V.2.7)) show that (V.2.1) holds.

As we shall see from later sections concerned with the practical use of the Rayleigh quotient, the accuracy of the eigenvalue evaluated by the Rayleigh quotient, is twice that of the corresponding eigenvector being used; i.e. assuming that the vector \( x \) is an approximation to the eigenvector \( x_i \) say,

\[
x = x_i + \varepsilon y
\]

then the Rayleigh quotient of \( x \) will yield an approximation of the corresponding eigenvalue \( \lambda_i \), where:

\[
u(x) = \lambda_i + O(\varepsilon^2)
\]

(V.2.9)

and the notation \( O(\varepsilon^2) \) indicates that if \( \varepsilon = O(\varepsilon^2) \), then for a constant "a" we have:

\[
|\varepsilon| \leq a \varepsilon^2
\]

(V.2.10)

For proof, we can use the definition of the Rayleigh quotient for the vector \( x \) and obtain:

\[
u(x) = \frac{u(x_1 + \varepsilon y)}{u(x_1 + \varepsilon y)} = \frac{(x_1^t + \varepsilon y^t)x_1 + \varepsilon y}{(x_1^t + \varepsilon y^t)(x_1 + \varepsilon y)} = \frac{x_1^t Ax_1 + \varepsilon y^t A x_1 + x_1^t A y + \varepsilon^2 y^t y}{x_1^t x_1 + \varepsilon y^t y + x_1^t y + \varepsilon^2 y^t y}
\]

(V.2.11)

Now, since the matrix \( A \) is symmetric, the above formula can be rewritten as:

\[
u(x) = \frac{x_1^t A x_1 + 2 \varepsilon x_1^t A y + \varepsilon^2 y^t y}{x_1^t x_1 + 2 \varepsilon x_1^t y + \varepsilon^2 y y}
\]

(V.2.12)
The error vector $y$, however, can be expressed as:

$$y = \sum_{j=1}^{n} a_j x_j$$

which shows that:

$$x_i^t Ay = 0 ,$$

and

$$y^t x_i = 0 .$$

Therefore, the expression (V.2.12) can be rewritten as:

$$\lambda_i + \varepsilon^2 \sum_{j=1}^{n} \frac{a_j^2 \lambda_j}{j \neq i}$$

which after the expansion of the denominator by the binomial theorem gives the result:

$$u(x) = u(x_1 + \varepsilon y) = \frac{\lambda_i + \varepsilon^2 \sum_{j=1}^{n} \frac{a_j^2 \lambda_j}{j \neq i}}{1 + \varepsilon^2 \sum_{j=1}^{n} \frac{a_j^2}{j \neq i}}$$

which proves that the accuracy of the eigenvalue $u(x)$ obtained from the corresponding Rayleigh quotient of the vector $x$ is twice as accurate as the vector itself.
V.3 S.O.R. METHOD FOR THE MINIMISATION (MAXIMISATION) OF THE RAYLEIGH QUOTIENT

In this section, we describe the S.O.R. method for the evaluation of the smallest and largest eigenvalues of a symmetric matrix \( A \), and their corresponding eigenvectors. In other words, we attempt to find the minimum and maximum values of \( \lambda \) for which

\[
Ax = \lambda x
\]

has a non-trivial solution \( x \).

The evaluation of the smallest eigenvalue of the given system has been shown by Ruhe [62-63] to be given by the minimum of the Rayleigh quotient. However, in addition, the largest eigenvalue can also be evaluated by a suitable choice of the initial guess of the eigenvector.

To formulate the process we assume that the \( n \) eigenvalues of \( A \) satisfy the following condition that:

\[
\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n
\]

where the multiplicities of the eigenvalues have been counted. We also split the coefficient matrix \( A \) in the following manner:

\[
A = D-L-U
\]

where \( L \) and \( U \) are the lower and upper triangular matrices respectively and \( D \) is a diagonal matrix. Therefore, according to the above splitting the matrix \( (A - \mu I) \) can be written as:

\[
C_i = A - \mu I = D_i - L - U
\]

where,

\[
D_i = D - \mu_i I
\]

An iterative eigenvalue procedure generates a sequence of vectors:

\[
x_1, x_2, \ldots, x_n, \ldots
\]

where:

\[
x_{i+1} = x_i - P_i
\]
and in the case of convergence the normalised vectors $x_i$ and the corresponding Rayleigh quotient will converge to an eigenvector and eigenvalue respectively whilst at the same time, the residual vectors $r_i$ defined as,

$$ r_i = r(x_i) = (A - wI)x_i $$

will converge to zero.

In the S.O.R. method, the above sequence of vectors are evaluated from the following formula:

$$ x_{i+1} = V_i^{-1}H_ix_i $$

where

$$ V_i = (D_i - \omega L) $$

and

$$ H_i = ((1 - \omega)D_i + \omega U). $$

It can be proved that if the starting vector $x_1$ is chosen such that the corresponding Rayleigh quotient, i.e.,

$$ \mu(x_1) = \frac{x_1^TAx_1}{x_1^Tx_1} $$

is less than the second smallest eigenvalue of $A$ and that $\omega$ is in the interval $(0,2)$, then the method will converge to the smallest eigenvalue of $A$, see Ruhe [62].

For the determination of the largest eigenvalue we need to choose the starting vector $x$ such that the corresponding Rayleigh quotient is greater than the second largest eigenvalue of $A$ with $\omega$ being in the interval $(0,2)$.

Note however, that in theory, the above conditions will ensure the convergence of the method to the smallest eigenvalue and the largest eigenvalue respectively, while in practice it was observed, for all the numerical examples, the following conditions were sufficient for the
convergence of the method if $0 < \omega < 2$:

i) a choice of the initial vector $x_1$ for which:

$$
\nu(x_1) \leq \min\{a_{i,i} \mid i=1,2,\ldots,n\} \quad \text{(V.3.11)}
$$

for the determination of the smallest eigenvalue and:

ii) a choice of the initial vector $x_1$ for which:

$$
\nu(x_1) \geq \max\{a_{i,i} \mid i=1,2,\ldots,n\} \quad \text{(V.3.12)}
$$

for the evaluation of the largest eigenvalue.

The rate of convergence of the S.O.R. method is governed by the second largest eigenvalue of the iteration matrix $V_1^\top H_1$ since its largest eigenvalue takes the value 1 as the method converges to an eigenvalue of $A$.

We now define $R$ as:

$$
R = \max \left\{ \frac{1}{\lambda_1} \left( V_1^\top H_1 \right) \right\} \quad \text{(V.3.13)}
$$

where $V_1$ and $H_1$ are the limiting value of the matrices $V_i$ and $H_i$ and are defined as:

$$
V_1 = ((D - \lambda_1 I) - \omega L) \quad \text{(V.3.14)}
$$

and

$$
H_1 = ((1 - \omega)(D - \lambda_1 I) + \omega U) \quad \text{(V.3.15)}
$$

with $\lambda_1$ being either the maximum or minimum eigenvalues of $A$. After the evaluation of one of these extreme eigenvalues of $A$ the algorithm (V.3.8) approaches the linear iteration,

$$
x_{i+1} = V_1^\top H_1 x_i \quad \text{(V.3.16)}
$$

in the limit from which a more accurate eigenvector is determined.

To obtain the fastest asymptotic rate of convergence in the S.O.R. method we have to choose $\omega$ such that $R$ is minimised. That is as in the
linear system case, see Young [58], we have to determine $\omega$ such that all the eigenvalues of the iteration matrix $V_k^{-1}H_k$ except the dominant one, $\lambda_1=1$, will have the same absolute value $|\omega-1|$.

The following theorem will define the best value of $\omega$ for the matrices with property A.

**THEOREM (V.3.1)**

Suppose that the system (V.3.8) has converged to the eigenvalue $\lambda_1$ of $A$ and that the matrix,

$$C_k = A - \lambda_1 I$$

(V.3.17)

has property A. Then, the eigenvalues $\eta$ of the iteration matrix $V_k^{-1}H_k$ and $\mu$ of the Jacobi iteration matrix $B$ defined as:

$$B = 1 - (D - \lambda_1 I)^{-\frac{1}{2}}C_k (D - \lambda_1 I)^{-\frac{1}{2}}$$

(V.3.18)

which are ordered as:

$$1 = \mu_1 > \mu_2 > \epsilon > \mu_3 > \ldots > \mu_n = -1$$

(V.3.19)

satisfy the relation:

$$(\eta - 1 + \omega)^2 = \omega^2 \mu_1^2$$

(V.3.20)

and if $\omega$ is chosen so that:

$$\omega = \omega_c = 2 / (1 + (1 - \mu_2^2)^\frac{1}{2})$$

(V.3.21)

we attain the best asymptotic rate of convergence $R$ defined by:

$$R = \omega_c^{-1} = [(1 - \mu_2^2)^\frac{1}{2}] / [1 + (1 - \mu_2^2)^\frac{1}{2}]$$

(V.3.22)

For the proof, see Ruhe [62].
V.4 DETERMINATION OF THE EXTREME EIGENVALUES BY A METHOD BASED ON A NEW SPLITTING OF THE MATRIX $C_i$.

In this section we seek the solution of the equation (V.3.1) by an iterative method where an alternative splitting to that used in the S.O.R. method is used. We first introduce the conditions under which the method is globally convergent and later on discuss two alternative strategies for increasing the rate of convergence.

V.4.1 General Theory

The basic iteration formula in the present method is the same as in (V.3.6). The consecutive vectors $x_1, x_2, \ldots, x_n, \ldots$ are computed according to the formula (V.3.8), i.e.,

$$x_{i+1} = V_i^{-1}H_i x_i$$

(V.4.1.1)

where the matrices $V_i$ and $H_i$ are now defined as:

$$V_i = (D_i - L)D_i^{-1}(D_i - U)$$

(V.4.1.2)

and

$$H_i = L D_i^{-1} U$$

(V.4.1.3)

with $L, U$ being the lower triangular and upper triangular components of the symmetric matrix $A$ respectively and $D_i$ is defined as:

$$D_i = D - \mu_i$$

(V.4.1.4)

where $\mu_i$ is the approximation to the eigenvalue of $A$ (smallest/largest) obtained by the evaluation of the Rayleigh quotient for the vector $x_i$.

The splitting of the matrix $C_i = A - \mu_i I$ can now be defined as:

$$C_i = A - \mu_i I = V_i - H_i$$

(V.4.1.5)

The vector $p_i$ defined in (V.3.6) and the residual vector $r_i$ given by
(V.3.7) are defined in terms of $V_i$ and $H_i$ as:

$$p_i = x_i - x_{i+1} = x_i - V_i^{-1}H_i x_i = V_i^{-1}(V_i - H_i) x_i = V_i^{-1}C_i x_i$$  \hspace{1cm} (V.4.1.6)

and,

$$r_i = (A - \mu_i I)x_i = C_i x_i = (V_i - H_i)x_i.$$  \hspace{1cm} (V.4.1.7)

We now attempt to give sufficient conditions under which the sequence $\mu_1, \mu_2, \ldots, \mu_i, \ldots$ defined as:

$$\mu_i = \frac{x_i^t A x_i}{x_i^t x_i}$$  \hspace{1cm} (V.4.1.8)

forms a monotone sequence (i.e., either an increasing sequence or a decreasing one), since this would guarantee that the sequence $\mu_i$ will converge to some value $\bar{\mu}$. Secondly, we study whether this $\bar{\mu}$ is an eigenvalue of (V.3.1) and finally, to show that in most cases, we have,

$$\bar{\mu} = \lambda_1 \quad \text{when } \mu_i \text{ is a decreasing sequence}$$  \hspace{1cm} (V.4.1.9)

and

$$\bar{\mu} = \lambda_n \quad " \quad \text{an increasing "}$$

**LEMMA (V.4.1.1):**

Let $x_i$ be the vectors evaluated by (V.3.8) and $\mu_i$ be the corresponding sequence computed by the Rayleigh quotient, then:

$$\mu_{i+1} - \mu_i = -p_i^t (V_i + H_i) p_i / x_i^t x_{i+1} x_{i+1}$$  \hspace{1cm} (V.4.1.10)

where $p_i$ is given by (V.4.1.6) and $V_i$ and $H_i$ are defined as (V.4.1.2) and (V.4.1.3) respectively.

**Proof:**

We calculate:

$$(\mu_{i+1} - \mu_i) x_{i+1}^t x_{i+1} = x_{i+1}^t x_{i+1} x_{i+1} - x_{i+1}^t x_{i+1} x_{i+1} - \mu_{i} x_{i+1}^t x_{i+1} x_{i+1}$$

$$= x_{i+1}^t x_{i+1} x_{i+1} - \mu_{i} x_{i+1}^t x_{i+1} x_{i+1}$$

$$= x_{i+1}^t A x_{i+1} - \mu_{i} x_{i+1} x_{i+1}.$$  \hspace{1cm} (V.4.1.11)
Since the matrix A is symmetric, then for any vectors x and y we have:

\[ x^t Ay = y^t Ax. \]  \hspace{1cm} (V.4.1.12)

Therefore, bearing in mind that \( x_{i+1}^t x_i = x_i^t x_{i+1} \) we can rewrite the formula (V.4.1.11) in the following form:

\[
\begin{align*}
(u_i + u_{i+1}) x_i^t x_{i+1} &= x_{i+1}^t A x_i - x_i^t A x_{i+1} - x_i^t A x_i + x_{i+1}^t A x_i = \\
&= x_{i+1}^t A x_i - x_i^t A x_{i+1} + u_i x_i^t x_{i+1} + x_{i+1}^t u_i x_i = \\
&= (x_{i+1}^t x_{i+1})^t((A - u_i I)x_i + (A - u_i I)x_{i+1}) = \\
&= -(x_{i+1}^t x_{i+1})^t((A - u_i I)x_i + (A - u_i I)x_{i+1}) = \\
&= -2p_i^t (A - u_i I)x_i + p_i^t (A - u_i I)(x_i - x_{i+1}) = \\
&= -2p_i^t f_i + p_i^t (A - u_i I)p_i. \hspace{1cm} (V.4.1.13)
\end{align*}
\]

We now show that \( r_i \) can be expressed as:

\[ r_i = V_i p_i. \hspace{1cm} (V.4.1.14) \]

For this we have:

\[
\begin{align*}
r_i &= (A - u_i I)x_i = (V_i - H_i) x_i = \\
&= V_i x_i - V_i H_i x_i = V_i x_i - V_i x_{i+1} = \\
&= V_i (x_i - x_{i+1}) = V_i p_i. \hspace{1cm} (V.4.1.15)
\end{align*}
\]

Hence, the expression (V.4.1.13) can be rewritten as:

\[
\begin{align*}
(u_i + u_{i+1}) x_i^t x_{i+1} &= -2p_i^t V_i p_i + p_i^t (A - u_i I)p_i = \\
&= -2p_i^t V_i p_i + p_i^t (V_i - H_i) p_i = \\
&= -p_i^t (V_i + H_i) p_i. \hspace{1cm} (V.4.1.16)
\end{align*}
\]

Therefore, \( u_i + u_{i+1} = -p_i^t (V_i + H_i) p_i / x_i^t x_{i+1} \)

and the relation (V.4.1.10) has been obtained.
From the expression (V.4.1.10) we can now formulate our procedure in order to obtain either of the extreme eigenvalues of A. That is, for $u_i$ to be a decreasing (increasing) sequence, the quantity $(u_{i+1} - u_i)$ should be a negative (positive) quantity. For this, it is evident that if the matrix $(V_i + H_i)$ is a positive (negative) definite matrix for all $i = 1, 2, \ldots$ then the sequence $u_i$ will form an decreasing (increasing) sequence.

To find out what behaviour the matrix $(V_i + H_i)$ for any $i$ would have, we use the following two Lemmas.

**LEMMA (V.4.1.2):**

Let $A$ be an $m \times n$ rectangular matrix and let

$$B = A^TA \quad (V.4.1.17)$$

then $B$ is symmetric positive semi-definite and if $\text{rank}(A) = n$, it is positive definite.

For the proof see Stewart [64].

**LEMMA (V.4.1.3):**

The matrices $V_i$ and $H_i$ as given in (V.4.1.2) and (V.4.1.3) are positive definite and positive semi-definite respectively, if $D_i$ has no negative elements and are negative definite and negative semi-definite if $D_i$ has no positive elements.

**Proof:** (for the positive definite case)

Suppose that we have:

$$d_{i,i} > 0 \quad i = 1, 2, \ldots, n \quad (V.4.1.18)$$

therefore, $D_i$ can be written as:

$$D_i = D_i^{-1/2}D_i^{1/2} \quad (V.4.1.19)$$

and we have

$$V_i = (D_i - L)D_i^{-1/2}D_i^{1/2}(D_i - U) = B^TB \quad (V.4.1.20)$$
where, \[ B = (D_i - I)D_i^{-1} \] \hfill (V.4.1.21)

The matrix \( H_i \) can also be written as:
\[ H_i = LD_i^{-1}D_i^{-1}U = C^TC \] \hfill (V.4.1.22)
where, \[ C = LD_i^{-1} \] \hfill (V.4.1.23)

The Lemma (V.4.1.2) together with relation (V.4.1.20) prove that the matrix \( V_i \) is a positive definite matrix. Furthermore because of the structure of the matrix \( H_i \) we have that \( \text{rank}(H_i) = n - 1 \), therefore, we can again use Lemma (V.4.1.2) and relation (V.4.1.22) to conclude that the matrix \( H_i \) is a positive semi-definite matrix.

From the above two Lemmas we can now prove the following theorem:

**THEOREM (V.4.1.1):**

If the starting vector \( x_1 \) in the iterative formula (V.3.8) is chosen such that the corresponding Rayleigh quotient is less than the minimum of the diagonal elements of the matrix \( A \), i.e.
\[ \mu_1 = \frac{x_1^TAx_1}{x_1^Tx_1} < \min_i a_{i,i}, \quad i=1,2,\ldots,n \] \hfill (V.4.1.24)
then the sequence \( \mu_i \) will form a decreasing sequence. Also if this vector is chosen such that:
\[ \mu_1 = \frac{x_1^TAx_1}{x_1^Tx_1} > \max_i a_{i,i}, \quad i=1,2,\ldots,n \] \hfill (V.4.1.25)
then the sequence \( \mu_i \) will form an increasing sequence and in both cases, the method described will converge in the following sense:
\[ \mu_i \rightarrow \bar{\mu} = \lambda_1 \] \hfill (V.4.1.26)
and
\[ r_i \rightarrow 0. \] \hfill (V.4.1.27)
Proof:

We prove the theorem for when the initial vector \( x_1 \) is chosen such that the relation (V.4.1.24) holds. The two Lemmas (V.4.1.2) and (V.4.1.3) imply that if,

\[
\nu_1 < \min a_{i,i}, \quad i=1,2,...,3
\]

then the two matrices \( V_i \) and \( H_i \) defined as in (V.4.1.2) and (V.4.1.3) are positive definite and positive semi-definite respectively. Therefore, the quantity on the right hand side of the expression (V.4.10) is negative and hence we have:

\[
\nu_{i+1} - \nu_i < 0. \tag{V.4.1.28}
\]

The above inequality clearly shows that with the condition (V.4.1.24) satisfied, then the sequence \( \nu_i \) will form a decreasing sequence.

However, the sequence \( \nu_i \) on the one hand is a decreasing sequence and on the other hand is bounded by the smallest eigenvalue of \( A \), hence it is convergent, i.e.

\[
\nu_i \to \overline{\nu} \tag{V.4.1.29}
\]

and

\[
\nu_i - \nu_{i+1} \to 0 \tag{V.4.1.30}
\]

Furthermore, since the matrices \( V_i \) and \( H_i \) are symmetric therefore, \( V_i + H_i \) is symmetric as well, hence we have;

\[
\kappa_{ci}(u_i - u_{i+1}) = p_i^T (V_i + H_i) p_i \geq \lambda_{\min} (V_i + H_i) p_i^T p_i \geq C \| p_i \|_2. \tag{V.4.1.31}
\]

where \( C \) represents a constant. From (V.4.1.30) and the above inequality it can easily be seen that \( \| p_i \|_2 \to 0 \) and since \( r_i = V_i p_i \) and \( \| V_i \| \) is bounded therefore,

\[
r_i \to 0 \tag{V.4.1.32}
\]

and the proof of the theorem is complete.

Note that, in theory, to ensure the convergence of the method to one of the extreme eigenvalues we need to have the following conditions for the selection of the initial vector \( x_1 \) as well as those given in
(V.4.1.24) and (V.4.1.25):

$$v_1 = \frac{x_1^tAx_1}{x_1^tx_1} < \lambda_2, \text{ if } \lambda_1 \text{ is required} \quad (V.4.1.33)$$

and

$$v_2 = \frac{x_1^tAx_1}{x_1^tx_1} > \lambda_{n-1}, \text{ if } \lambda_n \text{ is required} \quad (V.4.1.34)$$

however, in the numerical experiments described in the next chapter the conditions given in (V.4.1.24) and (V.4.1.25) were sufficient for the convergency of the method to the desired eigenvalue.

The numerical experiments carried out on the described method showed that the efficiency of the algorithm is comparable with the Gauss-Seidel iteration method. However, there are two alternative ways of speeding up the process of evaluating the eigenvalues and eigenvectors of a matrix $A$ by the proposed method. These alternatives are:

1) a shift of origin on the iteration matrix;

ii) introducing a preconditioning parameter $\omega$ into the method.

These two strategies are described in the next two sections.
V.4.2 Shift of Origin of the Iteration Matrix

The iteration matrix in our method is defined as:

\[ \mathcal{K}_1 = V_1^{-1} H_1 \]  \hspace{1cm} (V.4.2.1)

where \( V_1 \) and \( H_1 \) are the matrices defined in (V.4.1.2) and (V.4.1.3).

The above matrix has some important properties which enable us to find a suitable shifting parameter to speed up the process. These properties are given in the following Lemmas.

**Lemma (V.4.2.1)**

The iteration matrix \( \mathcal{K}_1 \) is similar to a positive semi-definite matrix.

**Proof:**

To prove the Lemma, we show that the matrix \( \mathcal{K}_1 \) defined as:

\[ \mathcal{K}_1 = Q_1 \mathcal{K}_1 Q_1^{-1} \]  \hspace{1cm} (V.4.2.2)

which is similar to \( \mathcal{K}_1 \) is a positive semi-definite matrix where:

\[ Q_1 = D_1^{-1} (D_1 - U) \]  \hspace{1cm} (V.4.2.3)

and hence:

\[ Q_1^{-1} = D_1 (D_1 - U)^{-1} D_1^{-1} \]  \hspace{1cm} (V.4.2.4)

Note that the matrix \( D_1^{-1} \) used in the above definitions does exist because the elements of the diagonal matrix \( D_1 \) are either all positive (when the smallest eigenvalue is being evaluated) or are all negative (when the largest eigenvalue is being determined) i.e., in the later case \( Q_1 \) is defined as:

\[ Q_1 = -(D_1 - U)(-D_1)^{\dagger} \]  \hspace{1cm} (V.4.2.5)

However, we continue the proof assuming that \( D_1 \) has positive elements. Substituting \( Q_1 \) defined in (V.4.2.3) into the expression (V.4.2.2) we have:
\[
\overline{K}_1 = Q_1 K_1 Q_1^{-1} = D_1^{-1} (D_1 - U) ((D_1 - L) D_1^{-1} (D_1 - U))^{-1} H_1 (D_1 - U)^{-1} D_1^{-1}
\]

\[
= D_1^{-1} (D_1 - U) (D_1 - U)^{-1} D_1 (D_1 - L)^{-1} H_1 (D_1 - U)^{-1} D_1^{-1}
\]

\[
= D_1^{-1} (D_1 - L)^{-1} H_1 (D_1 - U)^{-1} D_1^{-1}
\]  

(V.4.2.6)

Since  
\[
((D_1 - U)^{-1})^t = (D_1 - L)^{-1}
\]  

(V.4.2.7)

and  
\[
D_1^{-1} (D_1 - L)^{-1} = (((D_1 - U)^{-1} D_1^{-1})^t
\]  

(V.4.2.8)

we have:
\[
\overline{K}_1 = B_1 H_1 B_1^t
\]  

(V.4.2.9)

where
\[
B_1 = D_1^{-1} (D_1 - L)^{-1}
\]  

(V.4.2.10)

The matrix \( H_1 \), however, is a positive semi-definite matrix, hence we have:
\[
\overline{K}_1 = F_1 F_1^t
\]  

(V.4.2.11)

where:
\[
F_1 = B_1 H_1^{\frac{1}{2}}
\]  

(V.4.2.12)

The relation (V.4.2.11) together with Lemma (V.4.1.2) indicate that the matrix \( K_1 \) is a positive semi-definite matrix and the proof of the Lemma is complete.

Note that since \( K_1 \) is a singular matrix its smallest eigenvalue is equal to zero.

**Lemma (V.4.2.2):**

The largest eigenvalue of the iteration matrix \( K_1 \) converges to unity as the sequence \( \mu_1 \) converges to one of the eigenvalues of \( A \).

**Proof:**

The relation:
\[
x_{i+1} = V_i^{-1} H_1 x_i
\]  

(V.4.2.13)
illustrates that once the above system has converged, then it can be used
as the eigenvalue statement of the matrix $\mathcal{K}_1$ for $\lambda=1$ and corresponding
eigenvector $x_1$. On the other hand (V.4.2.14) represents the power method
for the matrix $\mathcal{K}_1$ and therefore the eigenvector $x_1$ will correspond to the
largest eigenvalue of $\mathcal{K}_1$.

From the above two Lemmas we can conclude that once the system (V.4.2.13)
has converged to an eigenvalue of $\lambda$ we have:

$$0 \leq \lambda_{\mathcal{K}_1} \leq 1.$$  \hspace{1cm} (V.4.2.14)

Now that the smallest eigenvalue of the iteration matrix $\mathcal{K}_1$ and
the limit of its largest eigenvalue have been determined, it is possible
to apply the "shift of origin" technique related to the power method for
the matrix $\mathcal{K}_1$ and reduce the number of iterations.

The rate of convergence of the system (V.4.2.13) is presently
governed by $(\lambda_{n-1}/\lambda_n)=\lambda_{n-1}$ where $\lambda_{n-1}$
is the second largest eigenvalue of $\mathcal{K}_1$. To reduce this value we choose the shifting parameter $\omega$ as:

$$\omega = (\lambda_{n-1}/2) - c.$$  \hspace{1cm} (V.4.2.15)

and the iteration formula (V.4.2.13) is now replaced by:

$$x_{i+1} = \mathcal{K}_1x_i - \omega x_i.$$  \hspace{1cm} (V.4.2.16)

Although, in practice, the value of $\lambda_{n-1}$ is generally unknown,
good estimates of $\lambda_{n-1}$ are possible as the size of the matrix $\mathcal{K}_1$ increases
which forces this value to become nearer to the largest eigenvalue of $\mathcal{K}_1$,
i.e. $\lambda_n=1$. In our numerical experiment it was observed that for $n>100$
the choices of $\omega=0.45$ to 0.49 can reduce the number of iterations required
for convergence to a factor of 2/3 or even 1/2.

Note that the definition of the shifting parameter $\omega$ by (V.4.2.15)
does not alter the position of the largest eigenvalue of $\mathcal{K}_1$ and its value
after the shift i.e. $\lambda_n = 1-\omega$
will be greater than $\lambda_1-\omega=-\omega$ in absolute value.
V.4.3 Introduction of the Preconditioning factor \( \omega \)

An alternative strategy for speeding up the process of determining the eigenvalues of \( A \) by the proposed method is to introduce an acceleration parameter into the iterative algorithm defined in (V.4.1.1) whereby the consecutive vectors \( x_1, x_2, \ldots, x_n \) are evaluated from a similar formula:

\[ x_{i+1} = V_i^{-1}H_i x_i \]

where now we have the matrices \( V_i \) and \( H_i \) defined as:

\[ V_i = (D_i - \omega L)^{-1}(D_i - \omega U) \]  \( \text{(V.4.3.1)} \)

and

\[ H_i = \omega^2 D_i^{-1}U^*(1-\omega)(L+U) \]  \( \text{(V.4.3.2)} \)

It can be easily seen from the two previous equations, that

\[ C_i = A - \mu_i I = V_i - H_i \]  \( \text{(V.4.3.2)} \)

In general, with the above splitting of the matrix \( C_i \), the matrix \( H_i \) is no longer a definite matrix, although the matrix \( V_i \) remains a positive (negative) definite matrix provided that the conditions (V.4.1.24) and (V.4.1.25) are valid respectively. In this case however, we can select a suitable range for the preconditioning parameter \( \omega \) such that for any value \( \omega \) in that range, the matrix summation \( V_i + H_i \) forms a definite matrix.

To determine the above range we have the following Lemma.

**Lemma (V.4.3.1)**

The matrix \( V_i + H_i \), where \( V_i \) and \( H_i \) are defined as in (V.4.3.1) and (V.4.3.2) is a positive definite matrix if \( \omega \) is chosen to be in the range

\[ \frac{2-\sqrt{2}}{2} \leq \omega \leq \frac{2+\sqrt{2}}{2} \]  \( \text{(V.4.3.3)} \)

and that the condition (V.4.1.24) is true for the initial vector \( x_1 \).

**Proof**

Assume that the vector \( x \) is an arbitrary vector. We evaluate the
quadratic form of the matrix \( V_i + H_i \) for the vector \( x \) as follows:

\[
x^t (V_i + H_i) x = x^t (D_i \omega L D_i^{-1} D_i \omega U + \omega^2 L D_i^{-1} U + (1-\omega)(L + U)) x
\]

\[
= x^t D_i x - (2\omega - 1)x^t (L + U) x + 2\omega^2 x^t L D_i^{-1} U x.
\]

(V.4.3.4)

Since \( \omega \) is in the range defined in (V.4.3.3) and that the matrix \( L D_i^{-1} U \) is a positive semi-definite matrix, see Lemma (V.4.1.3), therefore, it is easy to see that:

\[
(2\omega - 1)^2 x^t L D_i^{-1} U x \leq 2\omega^2 x^t L D_i^{-1} U x.
\]

(Hence, we have:

\[
x^t (V_i + H_i) x = x^t D_i x - (2\omega - 1)x^t (L + U) x + 2\omega^2 x^t L D_i^{-1} U x
\]

\[
\geq x^t D_i x - (2\omega - 1)x^t (L + U) x + (2\omega - 1)^2 x^t L D_i^{-1} U x
\]

\[
= x^t [(D_i - (2\omega - 1)L) D_i^{-1} (D_i - (2\omega - 1)U)] x
\]

\[
= x^t B_i^t B_i x > 0 \quad \text{(see Lemma (V.4.3.2))}
\]

(V.4.3.6)

where:

\[
B_i = (D_i - (2\omega - 1)L) D_i^{-1}
\]

(V.4.3.7)

and therefore, the proof of the Lemma is complete.

The following Lemma can similarly be proved for the matrix \( V_i + H_i \) to be negative definite.

**Lemma (V.4.3.2)**

The matrix \( (V_i + H_i) \), with \( V_i \) and \( H_i \) being the matrices defined in (V.4.3.1) and (V.4.3.2), is a negative definite matrix if \( \omega \) is in the range given by (V.4.3.3) and if the condition (V.4.1.25) holds for the starting vector \( x_i \).

It can be readily seen that once the matrix \( V_i + H_i \) is definite, then Theorem (V.4.1.1) is true for the global convergence of the preconditioned method.
However, it was shown, in section (V.2) that using the Rayleigh quotient of a matrix in an iterative process for the evaluation of eigenvalues, we get full machine accuracy in the eigenvalue after about half the number of iterations needed for the vectors to converge. Hence, it seems reasonable to say that the rate of convergence of the system (V.4.1.1) is determined by the convergence rate of the limiting iteration,

$$x_{i+1} = V^{-1}_H x_i,$$  \hspace{1cm} \text{(V.4.3.8)}

where $V$ and $H$ denote the limits of the matrices $V_i$ and $H_i$ respectively, i.e.,

$$V = (D_{1} - \omega L) D_{1}^{-1} (D_{1} - \omega U)$$

$$H = \omega^2 LD_{1}^{-1} U + (1-\omega)(L+U)$$ \hspace{1cm} \text{(V.4.3.9)}

where

$$D_{1} = D - \lambda I$$ \hspace{1cm} \text{(V.4.3.10)}

with $\lambda$ being one of the two extreme eigenvalues of $A$.

Assuming the coefficient matrix $A$ is positive definite, then the limit of the matrix $C_1 = A - \mu_1 I$, denoted by $C$, would be a positive semi-definite matrix if $\lambda_1$ is being evaluated and a negative semi-definite matrix if $\lambda_n$ is being determined. The limit of the iteration matrix $K_1$ defined as:

$$K_1 = ((D_{1} - \omega L) D_{1}^{-1} (D_{1} - \omega U))^{-1} \left( \omega^2 LD_{1}^{-1} U + (1-\omega)(L+U) \right)$$ \hspace{1cm} \text{(V.4.3.11)}

can then be expressed as:

$$K_1 = ((D_{1} - \omega L) D_{1}^{-1} (D_{1} - \omega U))^{-1} \left( \omega^2 LD_{1}^{-1} U + (1-\omega)(L+U) \right)$$

$$= ((D_{1} - \omega L) D_{1}^{-1} (D_{1} - \omega U))^{-1} ((D_{1} - \omega L) D_{1}^{-1} (D_{1} - \omega U) - D_{1} + L + U)$$

$$= I - ((D_{1} - \omega L) D_{1}^{-1} (D_{1} - \omega U))^{-1} C$$

$$= I - B_{\omega}.$$ \hspace{1cm} \text{(V.4.3.12)}

We define the matrix $B_{\omega}$ as the preconditioned matrix of the method.

The matrix $\bar{B}_{\omega}$ defined as;
\[ \overline{B}_\omega = pB_\omega p^{-1} \quad (V.4.3.13) \]
where
\[ p = D_x^T(D_x - \omega L) \quad (V.4.3.14) \]
is similar to \( B_\omega \) and by applying the same algebra as given in Lemma (V.4.2.1) it can be seen that the matrix \( \overline{B}_\omega \) and hence \( B_\omega \) is positive semi-definite. From this observation, we have that if \( \lambda_i \) and \( \mu_i \) are the eigenvalues of the matrix \( K_x \) and \( B_\omega \) respectively, then they are real and are related through the relationship,
\[ \lambda_i = 1 - \mu_i \quad (V.4.3.15) \]
where
\[ \mu_j > 0 \quad 1, j = 1, 2, \ldots, n \]

In order to determine good estimates of the preconditioning parameter \( \omega \) near the optimum we attempt, in the next section, to find reasonable bounds for the second largest eigenvalue of the iteration matrix and study the behaviour of this value with respect to \( \omega \).
V.5 DETERMINATION OF GOOD ESTIMATES OF THE PARAMETER \( \omega \)

If we assume that \( \lambda_i \) and \( \mu_i \) are the eigenvalues of the matrices \( \mathcal{K}_L \) and \( B_\omega \) respectively where:

\[
|\lambda_1| \leq |\lambda_2| \leq \cdots \leq |\lambda_n| \leq |\mu_1| \leq |\mu_2| \leq \cdots \leq |\mu_n|,
\]

then from (V.4.3.12) we have that the largest eigenvalue of the iteration matrix \( \mathcal{K}_L \), \( \lambda_n = 1 \), corresponds to the smallest eigenvalue of the matrix \( B_\omega \), \( \mu_1 = 0 \). In order to study the convergence behaviour of the preconditioned method, we have to examine the second largest eigenvalue of \( \mathcal{K}_L \) defined as:

\[
\lambda_{n-1} = \max\{|1-\mu_2|, |1-\mu_n|\}.
\]

since the largest eigenvalue of \( \mathcal{K}_L \) remains constant for any value of \( \omega \).

In this section, we attempt to modify a similar analysis described by MISSIRLIS [69], whereby the eigenvalues of the preconditioned matrix \( B_\omega \) are formulated in terms of certain easily calculated quantities related to the matrix \( C_L \).

**Lemma (V.5.1)**

If the coefficient matrix \( A \) is positive definite, then the eigenvalues of the positive semi-definite matrix \( B_\omega \) satisfy the following inequality,

\[
\mu_1 \leq \frac{1}{\omega(\omega-2)}.
\]

**Proof**

Suppose that \( \mu_i \) is an eigenvalue of \( B_\omega \) and \( x \) is the corresponding eigenvector, then we have

\[
B_\omega x = \mu_i x
\]

or

\[
((D_L^\omega L)^{-1}(D_L^\omega U))^{-1}C_L x = \mu_i x.
\]

Hence

\[
C_L x = \mu_i (D_L^\omega L)^{-1}(D_L^\omega U) x
\]
and multiplying both sides of the above equation by $x^t$ we get,

$$
\mu_i = \frac{x^t C L x}{x^t (D L - \omega L)^{-1} (D L - \omega U) x}
$$

The quantity in the denominator of the above formula can be rewritten as

$$
x^t D L x - \omega x^t (L + U) x + \omega^2 x^t L D L^{-1} U x = \omega (2 - \omega) x^t (L - L - U) x
$$

where

$$
\hat{B}_\omega = ((\omega - 1) D L - \omega L)^{-1} ((\omega - 1) D L - \omega U) x
$$

If we assume, for the present, that the smallest eigenvalue of $A$ is being determined, then it can be easily seen that the matrix $\hat{B}_\omega$ is a positive definite matrix, see Lemma (V.4.1.3), and when we substitute the quantity obtained in (V.5.8) in the denominator of (V.5.7) we have:

$$
\mu_i = \frac{x^t C L x}{\omega (2 - \omega) x^t C L x + x^t \hat{B}_\omega x} \tag{V.5.10}
$$

Assuming that $x^t C L x \neq 0$ then since $x^t \hat{B}_\omega x$ and $x^t C L x$ are positive quantities and that the parameter $\omega$ is in the interval $[\frac{2 - \sqrt{2}}{2}, \frac{2 + \sqrt{2}}{2}]$, we have $\omega (2 - \omega) > 0$ and

$$
\mu_i = \frac{x^t C L x}{\omega (2 - \omega) x^t C L x + x^t \hat{B}_\omega x} \leq \frac{x^t C L x}{\omega (2 - \omega) x^t C L x} = \frac{1}{\omega (2 - \omega)} \tag{V.5.11}
$$

hence the proof of the Lemma is complete when the smallest eigenvalue of $A$ is being evaluated. However, in the case for the largest eigenvalue, the matrix, $C_L$ and $\hat{B}_\omega$ can be shown to be negative semi-definite and negative definite respectively. Hence, multiplying both the numerator and denominator of (V.5.10) by a negative sign we would create the same situation under which the above Lemma was proved for the smallest eigenvalue of $A$. 
The above Lemma produces an upper bound for the eigenvalues of the preconditioned matrix $B_{\omega}$. However, in order to determine a lower bound for the second smallest eigenvalue of $B_{\omega}$ we shall now study in more detail, the behaviour of the Rayleigh quotient of the matrix $B_{\omega}$ defined in (V.5.7) with respect to the matrices $(L+U)$ and $LD^{-1}_L U$. That is if we assume that:

$$B = D^{-1}_L (L+U)$$  \hfill (V.5.12)

is the Jacobi iteration matrix of $C_L$, then assuming that the vector $x$ now corresponds to the second smallest eigenvalue of $B_{\omega}$ we have:

$$\mu_2 = \frac{x^T C_L x}{x^T D_L x - \omega x^T (L+U)x + \omega^2 x^T LD^{-1}_L U x} = \frac{x^T D_L x - x^T D_L B x}{x^T D_L x - \omega x^T D_L B x + \omega^2 x^T LD^{-1}_L U x}$$  \hfill (V.5.13)

Assuming that $x^T D_L x \neq 0$, we have

$$\mu_2 = \frac{1 - \hat{\alpha}(x)}{1 - \omega \hat{\alpha}(x) + \omega^2 \beta(x)} = f(\hat{\alpha}, \hat{\beta}, \omega),$$  \hfill (V.5.14)

where

$$\hat{\alpha}(x) = \frac{x^T D_L B x}{x^T D_L x}$$  \hfill (V.5.15)

and

$$\hat{\beta}(x) = \frac{x^T LD^{-1}_L U x}{x^T D_L x}$$  \hfill (V.5.16)

From (V.5.14) it is clear that the eigenvalue $\mu_2$ could be evaluated if its corresponding eigenvector was known. However, since this vector is generally unknown, we have to rely on bounds for $\hat{\alpha}(x)$ and $\hat{\beta}(x)$ in order to determine reasonable values for the second smallest eigenvalue of the matrix $B_{\omega}$.

We now attempt to find lower bounds for $\mu_2$ assuming that the coefficient matrix $A$ is symmetric positive definite and is consistently ordered. See Chapter II.
**LEMMA (V.5.2)**

If the symmetric, positive definite matrix $A$ is consistently ordered, then

$$S(B) < 1,$$ \hspace{1cm} (V.5.17)

where

$$B = I - D^{-1}A = D^{-1}(L+U).$$ \hspace{1cm} (V.5.18)

For the proof see YOUNG [58].

**LEMMA (V.5.3)**

If the eigenvalues of the matrix $B$ defined in (V.5.12) lie in the range,

$$\lambda_{\min}(B) = \rho \leq \lambda_B \leq p = \lambda_{\max}(B)$$ \hspace{1cm} (V.5.19)

and that $C$ is a positive definite matrix, then for an arbitrary vector $x$, we have:

i) $\rho \leq \hat{a}(x) \leq p$ \hspace{1cm} (V.5.20)

and

ii) $0 \leq \hat{b}(x) \leq S(\bar{L}U)$ \hspace{1cm} (V.5.21)

where:

$$\bar{L} = D_{\bar{L}}^{-1}LD_{\bar{L}}^{-1}$$ \hspace{1cm} (V.5.22)

and

$$\bar{U} = \bar{L}^T = D_{\bar{L}}^{-1}UD_{\bar{L}}^{-1}.$$ \hspace{1cm} (V.5.23)

**Proof**

For the proof of (i) we calculate

$$\hat{a}(x) = \frac{x^T D_{\bar{L}} B x}{x^T D_{\bar{L}} x} = \frac{x^T D_{\bar{L}}^4 (D_{\bar{L}}^4 B D_{\bar{L}}^{-4}) D_{\bar{L}}^4 x}{x^T D_{\bar{L}}^4 D_{\bar{L}}^4 x} = \frac{y^T (D_{\bar{L}}^4 B D_{\bar{L}}^{-4}) y}{y^T y}$$ \hspace{1cm} (V.5.24)

where the vector $y$ is defined as:

$$y = D_{\bar{L}}^4 x.$$ \hspace{1cm} (V.5.25)

Therefore, since the matrix $D_{\bar{L}}^4 B D_{\bar{L}}^{-4}$ is symmetric we have:

$$\lambda_{\min}(D_{\bar{L}}^4 B D_{\bar{L}}^{-4}) \leq \hat{a}(x) \leq \lambda_{\max}(D_{\bar{L}}^4 B D_{\bar{L}}^{-4}).$$ \hspace{1cm} (V.5.26)
However, since \( B \) is similar to the matrix \( D_L^\frac{1}{2}BD_L^{-\frac{1}{2}} \) we have:

\[
\lambda_{\min}(D_L^\frac{1}{2}BD_L^{-\frac{1}{2}}) = \lambda_{\min}(B) = \rho \tag{V.5.27}
\]

and

\[
\lambda_{\max}(D_L^\frac{1}{2}BD_L^{-\frac{1}{2}}) = \lambda_{\max}(B) = p \tag{V.5.28}
\]

hence,

\[
\rho \leq \hat{\alpha}(x) \leq p \tag{V.5.29}
\]

and the proof of (i) is complete.

For the proof of (ii) we calculate:

\[
\hat{\beta}(x) = \frac{x^tLD_L^{-1}UX}{x^tD_L^\frac{1}{2}LD_L^{-\frac{1}{2}}UD_L^\frac{1}{2}D_L^{-\frac{1}{2}}x} = \frac{x^tD_L^{-\frac{1}{2}}LD_L^{-\frac{1}{2}}UD_L^{-\frac{1}{2}}x}{x^tD_L^{-\frac{1}{2}}D_L^{-\frac{1}{2}}x} = \frac{y^tD_L^{-\frac{1}{2}}UD_L^{-\frac{1}{2}}y}{y^ty} \tag{V.5.30}
\]

where \( \tilde{L} \) and \( \tilde{U} \) are defined in (V.5.21) and (V.5.22) respectively. Since \( \tilde{L} \tilde{U} \) is a symmetric matrix we have:

\[
\lambda_{\min}(\tilde{L}\tilde{U}) \leq \hat{\beta}(x) \leq \lambda_{\max}(\tilde{L}\tilde{U}) \tag{V.5.31}
\]

However, since \( \tilde{L}\tilde{U} \) can be rewritten as

\[
\tilde{L}\tilde{U} = \hat{L}D_L^{-\frac{1}{2}} \hat{U} \tag{V.5.32}
\]

where

\[
\hat{L} = D_L^{-\frac{1}{2}}L \tag{V.5.33}
\]

and

\[
\hat{U} = UD_L^{-\frac{1}{2}} \tag{V.5.34}
\]

hence using Lemma (V.4.1.3), it can be seen that the matrix \( \tilde{L}\tilde{U} \) is a positive semi-definite matrix, therefore,

\[
\lambda_{\min}(\tilde{L}\tilde{U}) = 0 \tag{V.5.35}
\]

and we have

\[
0 \leq \hat{\beta}(x) \leq S(\tilde{L}\tilde{U}) \tag{V.5.36}
\]

and the proof of (ii) is complete.

Note that a similar Lemma can be proved for when the largest eigenvalue of \( A \) is being evaluated.

Since we assume that the matrix \( C_L \) is positive semi-definite and consistently ordered, therefore, using Lemma (V.5.2) we have,
\[ \lambda_{\text{max}}(D_x^4BD_x^{-4}) = \lambda_{\text{max}}(B) = \rho \leq 1. \]  
\hfill (V.5.37)

However, we have:

\[ \check{B} = D_x^4BD_x^{-4} = D_x^4D_x^{-4}(L+U)D_x^{-4} = D_x^{-4}(L+U)D_x^{-4} = \check{L} + \check{U} \]  
\hfill (V.5.38)

and

\[ S(B) = S(\check{B}) = S(\check{L} + \check{U}) = ||\check{L} + \check{U}|| \leq ||\check{L}|| + ||\check{U}|| \]
\[ = 2||\check{L}|| = 2\sqrt{S(\check{L}\check{U})} \leq 2\sqrt{\beta} \]  
\hfill (V.5.39)

where

\[ S(\check{L}\check{U}) \leq \bar{\beta} \]  
\hfill (V.5.40)

Therefore, it can be easily seen that:

\[ -\rho \leq 2\sqrt{\beta} \]  
\hfill (V.5.41)

and

\[ \rho \leq 2\sqrt{\beta} \]  
\hfill (V.5.42)

From the above two expressions and (V.5.37) we have:

\[ -2\sqrt{\beta} \leq \rho \leq 0 \leq \rho \leq \min(1,2\sqrt{\beta}). \]  
\hfill (V.5.43)

We are now able to find a lower bound for the second smallest eigenvalue of the preconditioned matrix \( B_{\omega} \). However, to find such a bound we assume that the value of \( \hat{a}(x) \) where \( x \) is the eigenvector corresponding to the second smallest eigenvalue of \( B_{\omega} \) is bounded by,

\[ \rho \leq \hat{a}(x) \leq \bar{p} \]  
\hfill (V.5.44)

where \( \bar{p} \) denotes the second largest eigenvalue of the Jacobi iteration matrix \( B \) defined as,

\[ \bar{p} = p - \varepsilon. \]  
\hfill (V.5.45)

The above assumption can be justified since the smallest eigenvalue of \( B_{\omega} \) is obtained when \( \hat{a}(x) \) reaches its maximum value \( p \) which for this case since \( C_2 \) is positive semi-definite, we have \( p = 1 \).

The problem is, therefore, to minimise the function \( f \) defined as,

\[ f(\hat{a}, \hat{\beta}, \omega) = \frac{1-\hat{a}}{1-\hat{a}+\omega \hat{\beta}} \]  
\hfill (V.5.46)

with respect to \( \hat{a} \) and \( \hat{\beta} \).

To do this, we calculate,
and since \( \omega > 0 \) and \( |\hat{a}| \leq 1 \) we have:

\[
\text{sign}[\frac{\partial}{\partial \beta} f(\hat{a}, \hat{\beta}, \omega)] < 0 ,
\]

which indicates that the function \( f(\omega, \hat{a}, \hat{\beta}) \) is a decreasing function of \( \hat{\beta} \). Therefore, considering that \( \hat{\beta} \leq \bar{\beta} \) our problem reduces to the minimisation of

\[
f(\hat{a}, \bar{\beta}, \omega) = \frac{1-\hat{a}}{1-\omega \hat{a} + \omega \bar{\beta}}
\]

with respect to \( \hat{a} \).

If we assume \( \omega \) to have a fixed value, we have,

\[
\frac{\partial}{\partial \hat{a}} f(\hat{a}, \bar{\beta}, \omega) = \frac{\omega - \frac{1}{2} \frac{1}{\bar{\beta}}}{1-\omega \hat{a} + \omega \bar{\beta}}
\]

and hence:

\[
\text{sign}[\frac{\partial}{\partial \hat{a}} f(\hat{a}, \bar{\beta}, \omega)] = \text{sign}(-\omega \frac{1}{2} \bar{\beta} + \omega - 1) .
\]

The above discussion can be summarised in the following theorem whereby a lower bound for \( \nu_2 \) is formulated.

**THEOREM (V.5.1)**

Let \( \bar{\beta}, \rho \) and \( \bar{\rho} \) be the values defined in (V.5.40), (V.5.41) and (V.5.45) respectively. Then, a lower bound on the second smallest eigenvalue of the matrix \( B_\omega \) is given by:

\[
\nu_2 \geq \begin{cases} 
\frac{1-\bar{\rho}}{1-\omega \bar{\rho} + \omega \bar{\beta}} & \text{if } \bar{\beta} \geq 1/4 \text{ or if } \bar{\beta} < 1/4 \text{ and } \omega < \omega^* \\
\frac{1-\rho}{1-\omega \rho + \omega \beta} & \text{if } \bar{\beta} < 1/4 \text{ and } \omega < \omega^*
\end{cases}
\]

where for \( \bar{\beta} < 1/4 \) we define \( \omega^* \) by

\[
\omega^* = \frac{2}{1+\sqrt{1-4\bar{\beta}}}
\]
The following table verifies the inequalities given in (V.5.5.1) with respect to \( \hat{\beta} \) and \( \omega \):

<table>
<thead>
<tr>
<th>( \hat{\beta} )-Domain</th>
<th>( \omega )-Domain</th>
<th>( \hat{\beta} \omega^2 - \omega + 1 )</th>
<th>( \mu_2 ) Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \hat{\beta} \geq 1/4 )</td>
<td>( 0 &lt; \omega &lt; 2 )</td>
<td>( &gt; 0 )</td>
<td>( \phi_1(\omega) = \phi(\omega, \hat{\beta}, \bar{\beta}) )</td>
</tr>
<tr>
<td>( 0 &lt; \hat{\beta} \leq 1/4 )</td>
<td>( 0 &lt; \omega &lt; \omega^* )</td>
<td>( &gt; 0 )</td>
<td>( \phi_1(\omega) = \phi(\omega, \hat{\beta}, \bar{\beta}) )</td>
</tr>
<tr>
<td></td>
<td>( \omega = \omega^* )</td>
<td>( = 0 )</td>
<td>( \phi_2(\omega) = \phi(\omega, \hat{\alpha}, \bar{\beta}) )</td>
</tr>
<tr>
<td></td>
<td>( \omega^* &lt; \omega &lt; 2 )</td>
<td>( &lt; 0 )</td>
<td>( \phi_2(\omega) = \phi(\omega, \rho, \bar{\beta}) )</td>
</tr>
</tbody>
</table>

**TABLE (V.5.1)**

We now turn our attention to examine the behaviour of the function \( g \) defined as:

\[
g(\hat{\alpha}, \hat{\beta}, \omega) = 1 - f(\hat{\alpha}, \hat{\beta}, \omega)
\]

with respect to \( \omega \) in the range \((0, 2)\). We calculate,

\[
\frac{\partial}{\partial \omega} g(\hat{\alpha}, \hat{\beta}, \omega) = \frac{2\omega \hat{\beta} - \hat{\alpha}}{(1-\omega \hat{\beta} + \omega^2 \hat{\beta})^2}
\]

and therefore,

\[
\text{sign}\left[\frac{\partial}{\partial \omega} g(\hat{\alpha}, \hat{\beta}, \omega)\right] = \text{sign}(2\omega \hat{\beta} - \hat{\alpha})
\]

If we define \( h \) as

\[
h(\hat{\alpha}, \hat{\beta}, \omega) = 2\omega \hat{\beta} - \hat{\alpha}
\]

then the behaviour of the function \( g \) can be illustrated by the following table.

<table>
<thead>
<tr>
<th>( \omega )-Domain</th>
<th>( h(\hat{\alpha}, \hat{\beta}, \omega) )</th>
<th>( g(\hat{\alpha}, \hat{\beta}, \omega) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \omega &gt; \frac{\hat{\alpha}}{2\hat{\beta}} )</td>
<td>( &gt; 0 )</td>
<td>Increasing</td>
</tr>
<tr>
<td>( \omega = \frac{\hat{\alpha}}{2\hat{\beta}} )</td>
<td>( = 0 )</td>
<td>Stationary</td>
</tr>
<tr>
<td>( \omega &lt; \frac{\hat{\alpha}}{2\hat{\beta}} )</td>
<td>( &lt; 0 )</td>
<td>Decreasing</td>
</tr>
</tbody>
</table>

**TABLE (V.5.2)**
Using the above table we can determine the behaviour of the functions:

\[ \theta_1(\omega) = g(\bar{p}, \bar{\beta}, \omega) \]  
\[ \theta_2(\omega) = g(\rho, \bar{\beta}, \omega) \]

and with respect to the preconditioned parameter \( \omega \). The behaviour of the function \( \theta_1(\omega) \) is summarised in the accompanying table, where \( \omega^- \) is defined as:

\[ \omega^- = \frac{\bar{p}}{2\bar{\beta}} \]

<table>
<thead>
<tr>
<th>( \omega^- )-Domain</th>
<th>( h(\hat{\alpha}, \bar{\beta}, \omega) )</th>
<th>( \theta_1(\omega) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0&lt;\omega&lt;\omega^- )</td>
<td>&lt;0</td>
<td>Decreasing</td>
</tr>
<tr>
<td>( \omega=\omega^- )</td>
<td>=0</td>
<td>Stationary</td>
</tr>
<tr>
<td>( \omega^-&gt;\omega^- )</td>
<td>&gt;0</td>
<td>Increasing</td>
</tr>
</tbody>
</table>

**TABLE (V.5.3)**

The function \( \theta_2(\omega) \) is, however, an increasing function of \( \omega \) in the interval \( (0,2) \) and it can be seen that \( \theta_2(\omega)=0 \) when:

\[ \omega = \hat{\omega} = 2/(1+\sqrt{1-4\bar{\beta}/\rho}) \]

and since \( \text{sign}(\theta_2(0) \cdot \theta_2(1)) < 0 \) we have that \( 0<\hat{\omega}<1 \).

From the analysis so far and the tables (V.5.1),(V.5.3) we can summarise the relation between the functions \( \theta_1(\omega) \) and \( \theta_2(\omega) \) with respect to \( \omega \) as follows:

i) if \( \hat{\beta}>1/4 \), then for all values \( \omega \) in the interval \( (0,2) \) the function \( \theta_1(\omega) \) will dominate the function \( \theta_2(\omega) \) i.e. \( \theta_1(\omega)>\theta_2(\omega) \),

ii) if \( 0<\hat{\beta}<1/4 \), then for the values of \( \omega \) which lie in the range
(0, \omega^*) we have \theta_1(\omega) \geq \theta_2(\omega) and for the values of \omega in the interval (\omega^*, 2) we have \theta_2(\omega) \geq \theta_1(\omega).

These cases are further analysed in the following graphs.
FIGURE (V.5.1): Relation between $\theta_1(\omega)$ and $\theta_2(\omega)$ with respect to $\omega$ when $0 < \omega < 2$ and $\beta > 1/4$: $\theta_1(\omega) > \theta_2(\omega)$
FIGURE (V.5.2): Relation between $\theta_1(\omega)$ and $\theta_2(\omega)$ with respect to $\omega$ when $\bar{\omega} \leq 1/4$ and $\omega_p < \omega^*$:

if $0 < \omega < \omega^*$ then $\theta_1(\omega) > \theta_2(\omega)$, and

if $0 < \omega^* < \omega$ then $\theta_2(\omega) > \theta_1(\omega)$
FIGURE (V.5.3): Relation between $\theta_1(\omega)$ and $\theta_2(\omega)$ with respect to $\omega$ when $\frac{1}{2} \leq 1/4$ and $\omega_p > \omega^*$:

if $0 < \omega < \omega^*$ then $\theta_2(\omega) > \theta_1(\omega)$ and

if $0 < \omega < \omega^*$ then $\theta_1(\omega) > \theta_2(\omega)$
We now have to consider the possibilities described in Figures (V.5.1) (V.5.2) and (V.5.3) in order to study the rate of convergence of the preconditioned method which is governed by the second largest eigenvalue of the iteration matrix $\mathbf{K}_2$ in absolute value and to introduce a good estimate of the preconditioning parameter $\omega$.

If we consider case (i) illustrated in Figure (V.5.1) then it can be readily seen that $\theta_1(\omega)$ dominates over $\theta_2(\omega)$ for all values of $\omega$ in the interval $(0,2)$ and a good value of $\omega$ which minimises the bound for the second largest eigenvalue of $\mathbf{K}_2$ is obtained from the following quartic equation:

\[ 1 - \frac{1-\overline{\beta}}{1-\omega\overline{p}+\omega^2\overline{\beta}} = \frac{1}{\omega(2-\omega)} - 1 \]  
\[ (V.5.62) \]

or

\[ y(\omega) = 2\overline{\beta}\omega^4 - 2(\overline{p}+2\overline{\beta})\omega^3 + (1+5\overline{p}+\overline{\beta})\omega^2 - (3\overline{p}+2)\omega + 1 = 0 \]  
\[ (V.5.63) \]

It can be easily seen that:

\[ y(0) = 1 > 0, \quad y(1) = -\overline{\beta} < 0 \]

and

\[ y(1 + \sqrt{2}/2) \approx 0.0241 \overline{p} - 0.5095 \overline{p} + 0.5041 < 0, 0.5101 - 0.5095 \overline{p} > 0, \]

therefore, since,

\[ \text{sign}[y(0) \times y(1)] < 0 \]

and

\[ \text{sign}[y(1) \times y(1 + \sqrt{2}/2)] < 0 \]

we can conclude that $y(\omega)$ has at least two positive roots $\omega_1$ and $\omega_2$ such that

\[ 0 < \omega_1 < 1 \]  
\[ (V.5.64) \]

and

\[ 1 < \omega_2 < 1 + \sqrt{2}/2 \]  
\[ (V.5.65) \]

Furthermore, the inequality (V.5.65) shows that the two functions $\theta_1(\omega)$ defined in (V.5.58) and $\theta_3(\omega)$ defined as,

\[ \theta_3(\omega) = \frac{1}{\omega(2-\omega)} - 1 \]  
\[ (V.5.66) \]

have a common point when $\omega = \omega_2$. Therefore, considering the two possibilities
shown in Figures (V.5.4) and (V.5.5) we can define the value of \( \omega \) which minimises the bound \( S_2(\mathcal{K}) \) for the second largest eigenvalue of the matrix \( \mathcal{K} \) in the case where \( \beta \leq 1/4 \) as follows:

\[
\omega = \min \left\{ \omega_2, \frac{\bar{\omega}}{2\beta} \right\} \tag{V.5.67}
\]

and the corresponding bounds for \( S_2(\mathcal{K}) \) can be seen to be:

\[
S_2(\mathcal{K}) \leq \begin{cases} 
\frac{1}{\omega_2(2-\omega_2)} - 1 & \text{if } \omega = \omega_2 \\
\frac{4\bar{\omega}p-p^2}{(4\beta-p^2)} & \text{if } \omega = \frac{\bar{\omega}}{2\beta}
\end{cases} \tag{V.5.68}
\]

In the second case, when \( \beta \leq 1/4 \) then we have to see whether the evaluated value of \( \omega \) defined by (V.5.66) is less than \( \omega^* \) or not. Since this cannot be distinguished unless we solve (V.5.63) we shall study this evaluated value of \( \omega \) in the following manner. If

\[
\omega^* \geq 1 + \frac{\sqrt{2}}{2} \tag{V.5.69}
\]

which implies that, see (V.5.53)

\[
\frac{2}{1+\sqrt{1-4\beta}} \geq 1 + \frac{\sqrt{2}}{2} \\
or \\
\beta \geq 0.2426 \tag{V.5.70}
\]

then, bearing in mind that:

\[
y(1+\sqrt{2}/2) \geq 0.0241\bar{\beta}-0.5095\bar{p}+0.5041\geq 0.50995-0.5095\bar{p}>0
\]

which implies that \( \omega_2 \) is again in the range \( (1,1+\sqrt{2}/2) \), therefore, we take \( \omega \) as defined in (V.5.67) to be the preconditioned parameter and the bounds for \( S_2(\mathcal{K}) \) will be the same as before in (V.5.68). Otherwise if \( \beta < 0.2426 \) and \( \omega^* < \omega_2 \) then we choose \( \omega \) to be:

\[
\omega = \min \left\{ \frac{\bar{\omega}}{2\beta}, \omega^* \right\} \tag{V.5.71}
\]
Finally, the bounds for $S_2(\mathcal{H}_l)$, in this case we have that if $\omega = \frac{\bar{\beta}}{2\bar{\beta}}$, then $S_2(\mathcal{H}_l)$ is bounded by the second quantity in (V.5.68) and if $\omega = \omega^*$ then $S_2(\mathcal{H}_l)$ can be defined as

$$S_2(\mathcal{H}_l) \leq 1 - \frac{1 - \beta}{1 - \omega^* \beta \omega^*} \quad , \quad \beta < 0.2426 , \quad \omega = \omega^* < \omega_2.$$  \hspace{1cm} (V.5.72)

However, it can be easily seen that since

$$\text{sign}(\beta \omega^* - \omega^* + 1) < 0 \quad \text{if } \omega^* \leq \omega_2$$

then we can simplify the above as follows:

$$S_2(\mathcal{H}_l) \leq \frac{\omega^* - 1}{\omega^*}.$$  \hspace{1cm} (V.5.73)

The Figures (V.5.6) and (V.5.7) illustrate the bound $S_2(\mathcal{H}_l)$ for the case when $\beta \leq 1/4$ and $\omega^* < \omega_2$. 
FIGURE (V.5.4): Behaviour of $S_2(\mathcal{H}_z)$ as a function of $\omega$ when $\bar{\beta} \geq 1/4$ and $\omega_p < \omega_2$. 
FIGURE (V.5.5): Behaviour of $S_2(K_x)$ as a function of $\omega$ when $\theta \geq 1/4$ and $\omega_p > \omega_2$
FIGURE (V.5.6): Behaviour of $\theta_2(\mathcal{H}_\omega)$ as a function of $\omega$ when $\bar{b} < 0.2426$ and $\omega_p > \omega^*$.
FIGURE (V.5.7): Behaviour of $S_2(\mathcal{H}_x)$ as a function of $\omega$
when $\bar{\beta} < 0.2426$ and $\omega_p < \omega^*$
CHAPTER VI

NUMERICAL EXPERIMENTS ON ITERATIVE METHODS
VI.1 INTRODUCTION

In this chapter, we consider the iterative methods described in the previous chapter i.e. S.O.R. and the preconditioned methods and investigate their behaviour in practice. For this we have chosen 4 matrix examples for which one of the extreme eigenvalues and its corresponding eigenvector are determined with respect to different values of $\omega$. For each example we have the comparison of the number of iterations and the time required for evaluating the eigenvalues in both methods. Once the eigenvalue has been found then the evaluation of the Rayleigh quotient is no longer required and the process is continued without computing this quantity. We then compare the total number of iterations and the time required in both methods for evaluating a more accurate eigenvector.

An alternative iterative formula to that of (V.4.3.8) for computing the consecutive vectors $x_1$ is now proposed which reduces the amount of work required for the preconditioned method to converge to an eigenvector.

The main programs and all the routines used in our numerical experiments have been written in such a way that only the non-zero elements of the coefficient matrix $A$ need to be stored and operated on. These are given in Appendix II. The storage scheme used in our programs for the storage of the matrix $A$ requires 3 one-dimensional arrays $a(m), Ja(m), Ia(n+1)$ where $m$ indicates the number of the non-zero elements in the matrix and $n$ denotes the order of the matrix.
VI.2 THE INITIALISATION OF THE STARTING VECTOR

As shown in the previous chapter, in order to obtain the desired eigenvalue from the two extreme eigenvalues of $A$, the starting vector $x$ must be constructed such that one of the conditions (V.4.1.24) or (V.4.1.25) is true. However, the initialisation of this vector is not as complicated as it may seem. Indeed, the following strategy, requiring only a few operations, used in our numerical experiments worked perfectly for all cases.

Suppose that $a_{i,i}$ is the smallest diagonal element of $A$ and that there exists another non-zero element $a_{i,j}$ in the $i^{th}$ row of $A$. The initial eigenvector $x$ is now defined as,

$$x = [0, \ldots, 0, x_i, 0, \ldots, 0, x_j = 1, 0, \ldots, 0]^t$$  \hspace{1cm} (VI.2.1)

where $x_i$ can be any value which satisfies the following inequality:

$$\frac{a_{i,i}x_i^2 + 2a_{i,j}x_i + a_{j,j}}{1 + x_i^2} \leq a_{i,i}$$  \hspace{1cm} (VI.2.2)

or equivalently:

$$x_i \leq \frac{a_{i,i} - a_{j,j}}{2a_{i,j}}$$  \hspace{1cm} (VI.2.3)

However, to let other non-zero elements of the matrix $A$ contribute in the first step of the iterative process we may choose the vector $x$ to be:

$$x = [\epsilon, \ldots, \epsilon, x_i, \epsilon, \ldots, t, x_j = 1, \epsilon, \ldots, \epsilon]^t$$  \hspace{1cm} (VI.2.4)

where $\epsilon = 0.00001$ (say).  \hspace{1cm} (VI.2.5)

The above vector together with the initial value of the eigenvalue $\lambda$ defined as:

$$\lambda = \frac{a_{i,i}x_i^2 + 2a_{i,j}x_i + a_{j,j}}{1 + x_i^2}$$  \hspace{1cm} (VI.2.6)

will be suitable for computing the smallest eigenvalue of $A$. For the determination of the largest eigenvalue of $A$, however, the element $x_i$ of the starting vector $x$ should be one of the values satisfying the
inequality,

\[ x_i \geq \frac{a_{i,i} - a_{i,j}}{2a_{i,j}} \]  

(IV.2.7)

where in this case, \( a_{i,j} \) denotes the largest diagonal element of \( A \).
VI.3 NUMERICAL EXAMPLES

Example VI.3.1

Consider the matrix $A_1$ defined as:

$$A_1 = \begin{bmatrix}
4I_n & -X & & & \\
-X & 4I_n & -X & & \\
& -X & 4I_n & -X & \\
& & -X & 4I_n & \\
& & & & \\
0 & & & & \\
& 0 & & & \\
& & 0 & & \\
& & & 0 & \\
& & & & 0
\end{bmatrix}$$  

where

$$A_1 = \begin{bmatrix}
0 & 1 & & & & \\
1 & 0 & 1 & & & \\
& 1 & 0 & 1 & & \\
& & 1 & 0 & 1 & \\
& & & 1 & 0 & \\
& & & & 0 & \\
& & & & & 0
\end{bmatrix}$$  

The above matrix is derived from the finite difference discretisation of the Laplace partial differential equation:

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0.$$  

on a square region with specified (Dirichlet) boundary condition. A regular grid of size $h$ is chosen and the partial difference operator is replaced by a finite difference approximation derived from the method of squares and depicted by the following stencil.
The eigenvalues of the matrix $A_1$ are given by:

$$\lambda_{k,j} = 4(1 + \cos k \cos j \theta), \quad k,j = 1,2,\ldots,n$$

where $\theta = \pi/(n+1)$, see MARIVIN [51].

The largest and smallest eigenvalues and the corresponding eigenvectors of the above matrix $A_1$ of order (324*324), i.e., $N=18$ were obtained by both the S.O.R. and the proposed preconditioned methods and the eigenvalues are given in the following table:

<table>
<thead>
<tr>
<th>S.O.R. Method</th>
<th>Preconditioned Method</th>
<th>Theoretical Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1083665</td>
<td>0.1083659</td>
<td>0.1083655</td>
</tr>
<tr>
<td>7.8916342</td>
<td>7.8916341</td>
<td>7.8916344</td>
</tr>
</tbody>
</table>

**TABLE (VI.3.1)**

The Extreme Eigenvalues of $A_1$

The initial vector $x$ for the evaluation of the largest eigenvalue of $A_1$ was chosen to be

$$x = [1, \epsilon, \epsilon, \ldots, \epsilon, x_{20} = -1, \epsilon, \ldots, \epsilon]^t$$

where

$$\epsilon = 0.00001,$$

from which the initial value for the largest eigenvalue of $A_1$ can be calculated to be

$$\lambda^{(1)} = 5.0 > \max_{i} a_{i,i} = 4.$$

The Table (VI.3.2) illustrates the number of iterations and the time units, (where a unit of time on the ICL-1900 series at Loughborough University is roughly equal to half a second) required for the convergence of the method to the largest eigenvalue and the corresponding eigenvector of $A_1$ for different values of $\omega$. 
<table>
<thead>
<tr>
<th>The value of ( \lambda )</th>
<th>S.O.R. METHOD</th>
<th>PRECONDITIONED METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eigenvalue</td>
<td>Eigenvector</td>
</tr>
<tr>
<td></td>
<td>No. of</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Iterations</td>
<td>Total No. of</td>
</tr>
<tr>
<td></td>
<td>Time Units</td>
<td>iterations</td>
</tr>
<tr>
<td>1.0</td>
<td>78</td>
<td>126</td>
</tr>
<tr>
<td>1.1</td>
<td>68</td>
<td>112</td>
</tr>
<tr>
<td>1.2</td>
<td>59</td>
<td>97</td>
</tr>
<tr>
<td>1.3</td>
<td>52</td>
<td>86</td>
</tr>
<tr>
<td>1.4</td>
<td>46</td>
<td>76</td>
</tr>
<tr>
<td>1.5</td>
<td>41</td>
<td>68</td>
</tr>
<tr>
<td>1.6</td>
<td>39</td>
<td>65</td>
</tr>
<tr>
<td>1.7</td>
<td>40</td>
<td>66</td>
</tr>
<tr>
<td>1.8</td>
<td>47</td>
<td>78</td>
</tr>
<tr>
<td>1.9</td>
<td>78</td>
<td>129</td>
</tr>
</tbody>
</table>

**TABLE (VI.3.2)**

The Number of Iterations and Time Units Required to Obtain the Largest Eigenvalue and Corresponding Eigenvector of the Matrix \( A_1 \)
Example (VI.3.2)

In this example, we shall consider the matrix $A_2$ derived from the finite difference discretisation of the Poisson partial differential equation

$$\nabla^2 \phi = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = g(x,y)$$

on a square region with specified (Dirichlet) boundary conditions. A regular grid of mesh points of spacing $h$ is chosen and the partial difference operator $\nabla^2$ is replaced by an accurate finite difference approximation as given by the following computational molecule:

\[\begin{array}{c}
1 \\
\hline
& -16 \\
\hline
1 & h & -16 & h & 60 & h & -16 & h & 1 \\
\end{array}\]

The eigenvalues of $A_2$ as defined in (VI.3.3) are given by the following formula:

$$\lambda_{i,j} = 60 - t_i - t_j , \ i = 1,2,\ldots,n, \ j = 1,2,\ldots,n,$$

where:

$$t_k = 66 - (8 + 2\cos \frac{kw}{n+1})^2 , \ k = 1,2,\ldots,n.$$

The eigenvalues of the matrix $A_2$ of order $(225*225)$ evaluated from both the S.O.R. method and the preconditioned method are tabulated together with the theoretical solutions in Table (VI.3.3)
\( A_2 = \begin{pmatrix}
(B - I_n) & -16 I_n & I_n \\
-16 I_n & B & -16 I_n & I_n \\
I_n & -16 I_n & B & -16 I_n & I_n \\
I_n & -16 I_n & B & -16 I_n & I_n \\
I_n & -16 I_n & B & -16 I_n & I_n \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{pmatrix} \)

where

\( B = \begin{pmatrix}
59 & -16 & 1 \\
-16 & 60 & -16 & 1 \\
1 & -16 & 60 & -16 & 1 \\
0 & 1 & -16 & 60 & -16 \\
0 & 0 & 1 & -16 & 59 \\
\end{pmatrix} \)

\( n^2 + n \)

\[
\text{TABLE (VI.3.3)}
\]

The Extreme Eigenvalues of \( A_2 \)

<table>
<thead>
<tr>
<th>S.O.R. Method</th>
<th>Preconditioned Method</th>
<th>Theoretical solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9252607</td>
<td>0.9252610</td>
<td>0.9252602</td>
</tr>
<tr>
<td>126.465775</td>
<td>126.465775</td>
<td>126.465776</td>
</tr>
</tbody>
</table>
The initial vector $x$ used for the evaluation of the smallest eigenvalue of this matrix is as follows:

$$x = [1., 3., \epsilon, \epsilon, \ldots, \epsilon]^T$$

which yields an initial value for the smallest eigenvalue of $A_2$:

$$\lambda^{(1)} = 50.5 < \min_i \{a_{i, i}\} = 58.$$ 

The number of iterations and the computational costs of the two methods for evaluating the smallest eigenvalue of $A_2$ are compared in the Table (VI.3.4) for different values of $\omega$.

**Example VI.3.3**

Consider the matrix $A_3$ defined as:

$$
\begin{bmatrix}
-2 & -1 & 0 & \cdots & 0 \\
-1 & 2 & -1 & & \\
-1 & 2 & -1 & \ddots & \\
0 & \ddots & \ddots & \ddots & -1 \\
\end{bmatrix}
\begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
$$

which is derived from the finite difference discretisation of the following 2-point boundary value problem,

$$\frac{d^2 \phi}{dx^2} = f(x), \quad a \leq x \leq b$$

given the following boundary conditions:

$$\phi = f(a) \text{ at } x=a$$

and

$$\phi = f(b) \text{ at } x=b.$$
<table>
<thead>
<tr>
<th>The value of ( \lambda )</th>
<th>S.O.R. METHOD</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>PRECONDITIONED METHOD</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eigenvalue</td>
<td>Eigenvector</td>
<td>Eigenvalue</td>
<td>Eigenvector</td>
<td>Eigenvalue</td>
<td>Eigenvector</td>
<td>Eigenvalue</td>
<td>Eigenvector</td>
<td></td>
</tr>
<tr>
<td>No. of Iterations</td>
<td>Time Units</td>
<td>Total No. of iterations</td>
<td>Total Time Units</td>
<td>No. of Iterations</td>
<td>Time Units</td>
<td>Total No. of iterations</td>
<td>Total Time Units</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>161</td>
<td>238</td>
<td>211</td>
<td>262</td>
<td>70</td>
<td>183</td>
<td>97</td>
<td>219</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>138</td>
<td>206</td>
<td>180</td>
<td>226</td>
<td>62</td>
<td>162</td>
<td>86</td>
<td>195</td>
<td></td>
</tr>
<tr>
<td>1.2</td>
<td>118</td>
<td>176</td>
<td>155</td>
<td>195</td>
<td>48</td>
<td>125</td>
<td>69</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>102</td>
<td>152</td>
<td>132</td>
<td>167</td>
<td>43</td>
<td>112</td>
<td>60</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>1.4</td>
<td>88</td>
<td>132</td>
<td>113</td>
<td>145</td>
<td>36</td>
<td>94</td>
<td>50</td>
<td>114</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>77</td>
<td>115</td>
<td>97</td>
<td>126</td>
<td>30</td>
<td>79</td>
<td>41</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>69</td>
<td>103</td>
<td>83</td>
<td>111</td>
<td>27</td>
<td>70</td>
<td>36</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>1.7</td>
<td>65</td>
<td>98</td>
<td>75</td>
<td>105</td>
<td>78</td>
<td>204</td>
<td>230</td>
<td>401</td>
<td></td>
</tr>
<tr>
<td>1.8</td>
<td>72</td>
<td>108</td>
<td>87</td>
<td>117</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>1.9</td>
<td>113</td>
<td>170</td>
<td>130</td>
<td>179</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE (VI.3.4)**

The Number of Iterations and Time Units Required to Obtain the Smallest Eigenvalue and Corresponding Eigenvector of the Matrix \( A_2 \).
The simplest central difference operator:
\[
\frac{\partial^2}{\partial x^2} = \frac{1}{h^2} \begin{pmatrix}
-1 & h & -1
\end{pmatrix} + O(h^2)
\]

is now chosen to replace the partial differential equation on a grid of evenly spaced points along the x axis with spacing h and the resulting matrix $A_3$ is formed.

The analytical solution for the eigenvalues of the above matrix is given by:
\[
\lambda_k = 4 \sin^2 \frac{k\pi}{2(n+1)} , \quad k=1,2,\ldots,n.
\]

The extreme eigenvalues of the matrix $A_2$ of order $(200*200)$ evaluated from the S.O.R. method and the preconditioned methods together with the theoretical solutions are tabulated in Table (VI.3.5).

<table>
<thead>
<tr>
<th>S.O.R. METHOD</th>
<th>PRECONDITIONED METHOD</th>
<th>THEORETICAL SOLUTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000265</td>
<td>0.000264</td>
<td>0.000244</td>
</tr>
<tr>
<td>3.999734</td>
<td>3.999723</td>
<td>3.999756</td>
</tr>
</tbody>
</table>

**TABLE (VI.3.5)**

The Extreme Eigenvalues of $A_3$

The initial vector $\mathbf{x}$ used to evaluate the smallest eigenvalue of $A_3$ was chosen to be:
\[
\mathbf{x} = [e, e, \ldots, e, x_{n-1} = 1, x_n = 2]^t
\]
from which an initial value $\lambda^{(1)}$ is determined to be:
\[
\lambda^{(1)} = 1.2 < \min \{a_{i,i}\} = 2
\]

The number of iterations and the time required for the evaluation of
the smallest eigenvalue of $A_3$ and the corresponding eigenvector are given in Table (VI.3.6) which is followed by the first 100 elements of the eigenvector obtained from the preconditioned and the S.O.R. methods respectively.

As can be seen from Table (VI.3.6), although the preconditioned method is more efficient than the S.O.R. method for the determination of the eigenvalue, the total time required for evaluating the eigenvector by the preconditioned method is greater than that of the S.O.R. method. However we shall now introduce an alternative iterative formula to that shown in (V.4.1.1) whereby the amount of work involved for the determination of an accurate eigenvector by the proposed preconditioned method is reduced. In this alternative strategy, once the eigenvalue has converged after $k$ (say) iterations then the consecutive vectors $x_{k+1}, x_{k+2}, \ldots$ which are converging to the corresponding eigenvector are evaluated by the following formula:

\[ x_{i+1} = x_i - V_L^{-1} C_L x_i, \quad i = k+1, k+2, \ldots \]  

(VI.3.6)

where $V_L$ and $C_L$ are the matrices defined in (V.4.3.9).

Comparing the above formula with (VI.3.6) it can be easily seen that the number of multiplicative operations required for each step of the iteration process is reduced by $m$ where $m$ denotes the number of non-zero elements in the coefficient matrix $A$.

The matrix $A_3$ defined in (VI.3.5) was tested on this alternative formula and the results obtained are once again compared with the results of the S.O.R. method in Table (VI.3.6).
The Number of Iterations and Time Units Required to Obtain the Smallest Eigenvalue and Corresponding Eigenvector of the Matrix $A_3$

| The value of the eigenvalue | S.O.R. METHOD | | | Preconditioned Method | | |
|------------------------------|----------------|----------------|----------------|----------------|----------------|
|                              | Eigenvalue     | Eigenvector    |                  | Eigenvalue     | Eigenvector    |
|                              | No. of         | Time Units     | Total No. of     | No. of         | Time Units     |
|                              | Iterations     |                | iterations       | Iterations     |                |
|                              |                |                | Total Time Units |                |                |
| 1.87                         | 321            | 286            | 882              | 428            | 1.62           |
| 1.88                         | 315            | 281            | 777              | 413            | 1.63           |
| 1.89                         | 310            | 277            | 732              | 398            | 1.64           |
| 1.9                          | 305            | 273            | 688              | 383            | 1.65           |
| 1.91                         | 301            | 270            | 644              | 368            | 1.66           |
| 1.92                         | 299            | 269            | 601              | 355            | 1.67           |
| 1.93                         | 299            | 270            | 559              | 344            | 1.68           |
| 1.94                         | 302            | 272            | 517              | 334            | 1.69           |
| 1.95                         | 310            | 280            | 473              | 327            | 1.7            |
| 1.96                         | 326            | 294            | 500              | 345            |                |

TABLE (VI.3.6)
The First 100 Elements of the Eigenvector Corresponding to the Smallest Eigenvalue of $A_3$

obtained from the Preconditioned Method with $\varepsilon=10^{-6}$.

\[
\begin{array}{cccccccccccc}
0.0156483 & 0.0312778 & 0.0468997 & 0.0625102 & 0.0781054 & 0.0936815 \\
0.1092347 & 0.1247611 & 0.1402571 & 0.1557188 & 0.1711424 & 0.1865242 \\
0.2018604 & 0.2171472 & 0.2323810 & 0.2475580 & 0.2626744 & 0.2777266 \\
0.2927109 & 0.3076237 & 0.3224613 & 0.3372200 & 0.3518965 & 0.3664866 \\
0.3809873 & 0.3953949 & 0.4097058 & 0.4239166 & 0.4380237 & 0.4520238 \\
0.4659133 & 0.4796890 & 0.4933474 & 0.5068853 & 0.5202992 & 0.5335859 \\
0.5467423 & 0.5597649 & 0.5726508 & 0.5853967 & 0.5979995 & 0.6104561 \\
0.6227635 & 0.6349187 & 0.6469187 & 0.6587605 & 0.6704414 & 0.6819584 \\
0.6933087 & 0.7044896 & 0.7154983 & 0.7263321 & 0.7369883 & 0.7474645 \\
0.7577580 & 0.7678662 & 0.7777868 & 0.7875174 & 0.7970554 & 0.8063987 \\
0.8155449 & 0.8244917 & 0.8332371 & 0.8417789 & 0.8501149 & 0.8582432 \\
0.8661617 & 0.8738686 & 0.8813620 & 0.8886399 & 0.8957007 & 0.9025426 \\
0.9091640 & 0.9155632 & 0.9217387 & 0.9276840 & 0.9334126 & 0.9389081 \\
0.9441742 & 0.9492095 & 0.9540130 & 0.9585833 & 0.9629195 & 0.9670203 \\
0.9708849 & 0.9745123 & 0.9779016 & 0.9810519 & 0.9839626 & 0.9866329 \\
0.9890621 & 0.9912497 & 0.9931951 & 0.9948978 & 0.9963576 & 0.9975739 \\
0.9985465 & 0.9992751 & 0.9997597 & 1.0000000 & & \\
\end{array}
\]

For the other elements of this eigenvector, we have:

$$x_{n+i-1} = x_i \quad i=1,2,\ldots,n/2$$

The theoretical solution for the eigenvectors of the matrix $A_3$ is given by:

$$x_i = \left[ \sin \frac{i\pi}{(n+1)}, \sin \frac{2i\pi}{(n+1)}, \ldots, \sin \frac{ni\pi}{(n+1)} \right]^t.$$  

Max. Error Norm = $1.51 \times 10^{-4}$.  

The First 100 Elements of the Eigenvector Corresponding to the Smallest Eigenvalue of $A_3$

obtained from the S.O.R. Method with $\varepsilon=10^{-6}$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.014990</td>
<td>0.0299899</td>
<td>0.0449945</td>
<td>0.0600006</td>
<td>0.0750047</td>
</tr>
<tr>
<td>0.104918</td>
<td>0.1199674</td>
<td>0.1349262</td>
<td>0.1498646</td>
<td>0.1647787</td>
</tr>
<tr>
<td>0.1945198</td>
<td>0.2093395</td>
<td>0.2241202</td>
<td>0.2388586</td>
<td>0.2535507</td>
</tr>
<tr>
<td>0.2827822</td>
<td>0.2973142</td>
<td>0.3117856</td>
<td>0.3261928</td>
<td>0.3405323</td>
</tr>
<tr>
<td>0.3689936</td>
<td>0.3831083</td>
<td>0.3971411</td>
<td>0.4110885</td>
<td>0.4249469</td>
</tr>
<tr>
<td>0.4523830</td>
<td>0.4659539</td>
<td>0.4794220</td>
<td>0.4927842</td>
<td>0.5060369</td>
</tr>
<tr>
<td>0.5322007</td>
<td>0.5451053</td>
<td>0.5578872</td>
<td>0.5705433</td>
<td>0.5830704</td>
</tr>
<tr>
<td>0.6077248</td>
<td>0.6198459</td>
<td>0.6318254</td>
<td>0.6436604</td>
<td>0.6553477</td>
</tr>
<tr>
<td>0.6782678</td>
<td>0.6894946</td>
<td>0.7005620</td>
<td>0.7114674</td>
<td>0.722077</td>
</tr>
<tr>
<td>0.7431826</td>
<td>0.7534117</td>
<td>0.7634650</td>
<td>0.773399</td>
<td>0.7830338</td>
</tr>
<tr>
<td>0.8018689</td>
<td>0.8110051</td>
<td>0.8199505</td>
<td>0.8287028</td>
<td>0.8372596</td>
</tr>
<tr>
<td>0.8537782</td>
<td>0.8617355</td>
<td>0.8694886</td>
<td>0.8770355</td>
<td>0.8843742</td>
</tr>
<tr>
<td>0.8984190</td>
<td>0.9051214</td>
<td>0.9115080</td>
<td>0.9178770</td>
<td>0.9239267</td>
</tr>
<tr>
<td>0.9353617</td>
<td>0.9407438</td>
<td>0.9459003</td>
<td>0.9508298</td>
<td>0.9555308</td>
</tr>
<tr>
<td>0.9642421</td>
<td>0.9682500</td>
<td>0.9720244</td>
<td>0.9755641</td>
<td>0.9788683</td>
</tr>
<tr>
<td>0.9847656</td>
<td>0.9873570</td>
<td>0.9907991</td>
<td>0.9918211</td>
<td>0.9936923</td>
</tr>
<tr>
<td>0.9967096</td>
<td>0.9978546</td>
<td>0.9987566</td>
<td>0.9994150</td>
<td></td>
</tr>
</tbody>
</table>

The maximum error norm in the above vector is:

\[
\text{Max. Error Norm} = 1.49 \times 10^{-2}
\]
<table>
<thead>
<tr>
<th>The value of $\alpha$</th>
<th>S.O.R. METHOD</th>
<th>The value of $\alpha$</th>
<th>PRECONDITIONED METHOD AS DEFINED IN (VI.3.6)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total No. of iterations</td>
<td>Total Time Units</td>
<td>Total No. of iterations</td>
</tr>
<tr>
<td>1.87</td>
<td>882</td>
<td>428</td>
<td>1.61</td>
</tr>
<tr>
<td>1.88</td>
<td>777</td>
<td>413</td>
<td>1.62</td>
</tr>
<tr>
<td>1.89</td>
<td>732</td>
<td>398</td>
<td>1.63</td>
</tr>
<tr>
<td>1.9</td>
<td>688</td>
<td>383</td>
<td>1.64</td>
</tr>
<tr>
<td>1.91</td>
<td>644</td>
<td>368</td>
<td>1.65</td>
</tr>
<tr>
<td>1.92</td>
<td>601</td>
<td>355</td>
<td>1.66</td>
</tr>
<tr>
<td>1.93</td>
<td>559</td>
<td>344</td>
<td>1.67</td>
</tr>
<tr>
<td>1.94</td>
<td>517</td>
<td>334</td>
<td>1.68</td>
</tr>
<tr>
<td>1.95</td>
<td>473</td>
<td>327</td>
<td>1.69</td>
</tr>
<tr>
<td>1.96</td>
<td>500</td>
<td>345</td>
<td>-</td>
</tr>
</tbody>
</table>

**TABLE (VI.3.6)**

The Number of Iterations and Time Units Required for the S.O.R. Method and the Preconditioned Method as Defined in (VI.3.6) to Converge to the Smallest Eigenvalue of $A_{3}$
Example VI.3.4

A final experiment was carried out on the bounded matrix $A_4$ of order $(100 \times 100)$ as given below:

\[
A_4 = \begin{bmatrix}
5 & 2 & 1 & 1 \\
2 & 6 & 3 & 1 & 1 \\
1 & 1 & 3 & 6 & 3 & 1 & 1 \\
1 & 1 & 3 & 6 & 3 & 1 & 1 \\
1 & 1 & 2 & 5
\end{bmatrix}
\]

The extreme eigenvalues of this matrix were obtained from the two methods and these are tabulated in the following table:

<table>
<thead>
<tr>
<th>S.O.R. METHOD</th>
<th>PRECONDITIONED METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0077453</td>
<td>0.0077453</td>
</tr>
<tr>
<td>15.98451</td>
<td>15.98451</td>
</tr>
</tbody>
</table>

TABLE (VI.3.7)
The Extreme Eigenvalues of $A_4$

The initial vector $x$ used for the determination of the largest eigenvalue of $A_4$ was chosen to be:

\[ x = [x_1=1, x_2=\epsilon, \epsilon, \epsilon, \ldots] \]
which results in the initial value for the largest eigenvalue of $A_4$ to be:

$$\lambda^{(1)} = 6.88 > \max_{i} \{a_{i,i}\} = 6.$$ 

The above example was tested on the S.O.R. method and the preconditioned method as defined in (VI.3.6) and the number of iterations and the amount of time units required for the convergency of both methods are compared in Table (VI.3.8) for different values of $\omega$. 
| The Value of $\lambda$ | S.O.R. Method | | Preconditioned Method as Defined in (V1.3.6) |
|-----------------------|----------------|----------------|
|                        | Eigenvalue     | Eigenvector    | Eigenvalue     | Eigenvector    |
|                        | No. of Iterations | Time Units | Total No. of Iterations | Total Time Units | No. of Iterations | Time Units | Total No. of Iterations | Total Time Units |
| 1.0                    | 493            | 281          | 1036          | 390            | 169            | 182          | 349            | 241            |
| 1.1                    | 393            | 239          | 872           | 330            | 197            | 214          | 348            | 262            |
| 1.2                    | 334            | 204          | 731           | 279            | 119            | 129          | 243            | 169            |
| 1.3                    | 282            | 172          | 610           | 235            | 124            | 135          | 224            | 167            |
| 1.4                    | 238            | 145          | 504           | 196            | 97             | 105          | 174            | 130            |
| 1.5                    | 199            | 122          | 410           | 162            | 61             | 66           | 119            | 85             |
| 1.6                    | 165            | 101          | 326           | 132            | 47             | 51           | 88             | 63             |
| 1.7                    | 134            | 83           | 249           | 105            | 92             | 99           | 275            | 157            |
| 1.8                    | 110            | 68           | 177           | 81             | -              | -            | -              | -              |
| 1.9                    | 107            | 66           | 152           | 76             | -              | -            | -              | -              |

**TABLE (V1.3.8)**

The Number of Iterations and Time Units Required to Obtain the Largest Eigenvalue and Corresponding Eigenvector of the Matrix $A_4$
VI.4 CONCLUSION AND FINAL REMARKS

As can be seen from the numerical experiments carried out in the previous section the proposed preconditioned method has an overall less computational cost than the S.O.R. method. That is, although the amount of work (i.e. the number of multiplications and additions) involved in the preconditioned method at each step of the iteration process is nearly twice as much as that of the S.O.R., the number of iterations required for the convergency of the preconditioned method is such that the overall computational cost is less. In fact, the advantage of the preconditioned method over the S.O.R. is even greater when the eigenvalue is being determined. This is because the ratio of the amount of work while the Rayleigh quotient is being evaluated at each step is 3/2 and with the number of iterations required for the eigenvalue to be determined being less than half of that needed for the S.O.R. method to converge (based on the numerical experiments) we would need less time in the preconditioned method to obtain an eigenvalue. However, once the eigenvalue has been found and the evaluation of the Rayleigh quotient is terminated, then the ratio of the amount of work will be 2 and hence losing some of the time gained while the eigenvalue was being determined.

On the choice of a value of $\omega$ near optimal in the preconditioned method further research is needed to refine the bounds obtained for the second largest eigenvalue of the matrix $M_4$ described in the previous chapter. However, it should be mentioned here that for the matrix $A_3$ as defined in (VI.3.5) which is a consistently ordered matrix, the quartic equation (V.5.63) produced a very good choice of $\omega$. That is with $\beta=0.25006$ and $p=0.999975$ the value of $\omega$ evaluated from (V.5.63) can be seen to be $\omega=1.7$ and the results obtained for this value of $\omega$ are given in
in Table (VI.3.9) where \( \omega = 1.7 \) was used as the preconditioning parameter after the evaluation of the eigenvalue and comparing these results with those shown in Table (VI.3.7) it can be seen that the number of iterations is reduced. On the other hand, our experiments show that a value of \( \omega = 1.6 \) would be a good practical estimate of the preconditioning parameter for most moderate size matrices, i.e. \( n \geq 50 \).

<table>
<thead>
<tr>
<th>( \omega )</th>
<th>Total No. of iterations</th>
<th>Total Time Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.61</td>
<td>535</td>
<td>422</td>
</tr>
<tr>
<td>1.62</td>
<td>337</td>
<td>425</td>
</tr>
<tr>
<td>1.63</td>
<td>533</td>
<td>421</td>
</tr>
<tr>
<td>1.64</td>
<td>522</td>
<td>408</td>
</tr>
<tr>
<td>1.65</td>
<td>501</td>
<td>381</td>
</tr>
<tr>
<td>1.66</td>
<td>443</td>
<td>301</td>
</tr>
<tr>
<td>1.67</td>
<td>436</td>
<td>292</td>
</tr>
<tr>
<td>1.68</td>
<td>477</td>
<td>353</td>
</tr>
</tbody>
</table>

**TABLE (VI.3.9)**

The Number of Iterations and Time Units Required for the Preconditioned Method to Converge to the Smallest Eigenvalue and the Corresponding Eigenvector of the Matrix \( A_3 \) with \( \omega = 1.7 \) after the Determination of the Eigenvalue

Finally, these results confirm the conclusion that both strategies S.O.R. and the preconditioned method are effective methods for obtaining the extreme eigenvalues and the corresponding eigenvectors of large and sparse symmetric matrices.
CHAPTER VII

THE QUADRANT INTERLOCKING FACTORISATION (Q.I.F.) METHOD
VII1 INTRODUCTION
In the next two chapters, the subject of linear system solvers for parallel computers are studied. Our work is the extension of a factorisation method called the Quadrant Interlocking Factorisation (Q.I.F.) in which interlocking matrix quadrant components are considered instead of the standard LU triangular matrix factors, EVANS & HATZOPULOS [39]. All the methods described here are applicable to parallel computers of the type "Single Instruction stream Multiple Data stream (SIMD)" described in Chapter 1.

VII2 THE FACTORISATION PROCESS OF THE Q.I.F. METHOD
VII2.1 General Description
In this section we describe the Q.I.F. method for a coefficient matrix A in the system:
\[ Ax = b \]  
(VII.2.1.1)
In this method the coefficient matrix A is factorised into two matrices W and Z where
\[ A = W \cdot Z \]  
(VII.2.1.2)
and the matrices W and Z are of the forms as follows: if \( W_i \) and \( Z_i \) are the column vectors of the matrices W and \( Z^T \), then we have:
\[ W = \begin{bmatrix} W_1 & W_2 & \cdots & W_n \end{bmatrix} \]  
(VII.2.1.3)
\[ Z^T = \begin{bmatrix} Z_1 & Z_2 & \cdots & Z_n \end{bmatrix} \]
where the column vector \( W_i \)'s are of the following general form:
a) For n odd,
\[
W_i = \begin{cases} 
& [0 \ 0 \ \ldots \ 0 \ w_{i+1,i} \ \ldots \ w_{n-i,i} \ 0 \ \ldots \ 0]^T, \ i=1(1)\frac{n-1}{2} \\
& i \\
& [0 \ 0 \ \ldots \ 0 \ 1 \ 0 \ \ldots \ 0]^T, \ i=\frac{n+1}{2} \\
& i \\
& [0 \ 0 \ \ldots \ 0 \ w_{n-i+2,i} \ \ldots \ w_{i-1,i} \ 1 \ 0 \ \ldots \ 0]^T, \ i=\frac{n+3}{2}(1)n; \\
& n-i+1 
\end{cases}
\]  
(VII.2.1.4a)
b) For \( n \) even, 
\[
\begin{bmatrix}
0 & 0 & \ldots & 0 & w_{i+1,i} & \ldots & w_{n-1,i} & 0 & \ldots & 0
\end{bmatrix}^T, \ i=1(1)\frac{n}{2}-1
\]
\[
W_i = \begin{bmatrix}
0 & 0 & \ldots & 0 & 1 & 0 & \ldots & 0
\end{bmatrix}^T, \ i=\frac{n}{2}+1
\]  
(VII.2.1.4b)
\[
\begin{bmatrix}
0 & 0 & \ldots & 0 & w_{n-i+2,i} & \ldots & w_{i-1,i} & 1 & 0 & \ldots & 0
\end{bmatrix}^T, \ i=\frac{n}{2}+2(1)n
\]

Also, the column vector \( Z_i \)'s are of the following general form:

a) For \( n \) odd, 
\[
Z_i = \begin{bmatrix}
0 & 0 & \ldots & 0 & z_{i,i} & \ldots & z_{i,n-i+1} & 0 & \ldots & 0
\end{bmatrix}^T, \ i=1(1)\frac{n+1}{2}
\]  
(VII.2.1.5a)
\[
\begin{bmatrix}
0 & 0 & \ldots & 0 & z_{i,n-i+1} & \ldots & z_{i,i} & 0 & \ldots & 0
\end{bmatrix}^T, \ i=\frac{n+3}{2}(1)n
\]

b) For \( n \) even, 
\[
Z_i = \begin{bmatrix}
0 & 0 & \ldots & 0 & z_{i,i} & \ldots & z_{i,n-i+1} & 0 & \ldots & 0
\end{bmatrix}^T, \ i=1(1)\frac{n}{2}
\]  
(VII.2.1.5b)
\[
\begin{bmatrix}
0 & 0 & \ldots & 0 & z_{i,n-i+1} & \ldots & z_{i,i} & 0 & \ldots & 0
\end{bmatrix}^T, \ i=\frac{n+1}{2}(1)n
\]

For \( n=5 \), the matrices \( W \) and \( Z \) have the following forms,

\[
W = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix} \text{ 5*5}
\]

\[
Z = \begin{bmatrix}
z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} \\
z_{2,1} & z_{2,2} & z_{2,3} & z_{2,4} & 0 \\
z_{3,1} & z_{3,2} & z_{3,3} & 0 & 0 \\
z_{4,1} & z_{4,2} & z_{4,3} & z_{4,4} & 0 \\
z_{5,1} & z_{5,2} & z_{5,3} & z_{5,4} & z_{5,5}
\end{bmatrix} \text{ 5*5}
\]

Similarly for \( n=6 \), we have

\[
W = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \text{ 6*6}
\]

\[
Z = \begin{bmatrix}
z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} & z_{1,6} \\
z_{2,1} & z_{2,2} & z_{2,3} & z_{2,4} & z_{2,5} & 0 \\
z_{3,1} & z_{3,2} & z_{3,3} & z_{3,4} & 0 & 0 \\
z_{4,1} & z_{4,2} & z_{4,3} & z_{4,4} & z_{4,5} & 0 \\
z_{5,1} & z_{5,2} & z_{5,3} & z_{5,4} & z_{5,5} & z_{5,6} \\
z_{6,1} & z_{6,2} & z_{6,3} & z_{6,4} & z_{6,5} & z_{6,6}
\end{bmatrix}
\]
VII.2.2 The Computation of the Matrices W and Z

The elements of the matrices W and Z are evaluated in \( \left[ \frac{n-1}{2} \right] \) distinct stages where \( \left[ \frac{n-1}{2} \right] \) represents the largest integer number which is \( \leq\frac{n-1}{2} \).

From the given forms of the matrices W and Z in (VII.2.1.4) and (VII.2.1.5) and the equality (VII.2.1.2), it can easily be seen that the values of the elements of the first and last rows of the matrix Z are as follows:

\[
\begin{align*}
\text{and} \\
\begin{cases}
z_{1,i} &= a_{1,i} \\
z_{n,i} &= a_{n,i} \\
\end{cases}
\end{align*}
\]  

(VII.2.2.1)

The elements of the first and last columns of the matrix W are then evaluated by solving \((n-2)\) sets of \((2 \times 2)\) linear systems given by

\[
\begin{align*}
\begin{cases}
z_{1,i}w_{i,1} + z_{n,i}w_{n,1} &= a_{i,1} \\
z_{1,n}w_{i,1} + z_{n,n}w_{n,1} &= a_{i,n} \\
\end{cases}, i=2(1)n-1 & (VII.2.2.2)
\end{align*}
\]

This then completes the first stage of the factorisation process and in preparation for the next stage the elements of the matrix \(A\) are modified according to the following formula:

\[
a_{i,j} = a_{i,j} - w_{i,1}z_{1,j} - w_{n,1}z_{n,j}, i,j=2(1)n-1 & (VII.2.2.3)
\]

In general and at the \(i^{th}\) stage we have the relationships,

\[
\begin{align*}
\begin{cases}
z_{i,j} &= a_{i,j} \\
z_{n-i+1,j} &= a_{n-i+1,j} \\
\end{cases}, j=i(1)n-i+1 & (VII.2.2.4)
\end{align*}
\]

and

\[
Z = \begin{bmatrix}
z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} & z_{1,6} \\
0 & z_{2,2} & z_{2,3} & z_{2,4} & z_{2,5} & 0 \\
0 & 0 & z_{3,3} & z_{3,4} & 0 & 0 \\
0 & 0 & z_{4,3} & z_{4,4} & 0 & 0 \\
0 & z_{5,2} & z_{5,3} & z_{5,4} & z_{5,5} & 0 \\
z_{6,1} & z_{6,2} & z_{6,3} & z_{6,4} & z_{6,5} & z_{6,6}
\end{bmatrix}^{6 \times 6}
\]
and the solution of the $2 \times 2$ linear systems:

$$\begin{align*}
\begin{cases}
  z_{i,i}w_j + z_{n-i+1,i}w_j,n-i+1 &= a_{j,i}, \\
  z_{n-i+1,j}w_i + z_{j,n-i+1}w_j,n-i+1 &= a_{j,n-i+1},
\end{cases}
\end{align*}$$

(VII.2.2.5)

to give the unknown quantities $w_{j,i}$ for $j=i+1(1)n-i$, and finally the modified $a_{i,j}$'s are evaluated from the formula:

$$a_{k,l} = a_{k,l} - w_{k,i}z_{i,l} - w_{k,n-i+1}z_{n-i+1,l}, \quad k,l = i+1(1)n-i.$$  

(VII.2.2.6)

**VII.23 The Computational Cost of the Factorisation Process**

To analyse the computational cost of the factorisation process, we discuss two methods in which the ($2 \times 2$) linear systems in (VII.2.2.5) can be solved:

1. **Cramers Rule**: each system is solved according to the following algorithms:

   1.a. The evaluation of $T_1, T_2, T_3$ (say) defined as,

   $$T_1 = z_{i,i}z_{n-i+1,n-i+1} - z_{n-i+1,i}z_{i,n-i+1}$$
   $$T_2 = a_{j,i}z_{n-i+1,n-i+1} - a_{j,n-i+1}z_{n-i+1,i}$$
   $$T_3 = a_{j,i}z_{i,n-i+1} - a_{j,n-i+1}z_{i,i}$$

   1.b. Compute $w_{j,i} = T_2/T_1$ and $w_{j,n-i+1} = T_3/T_1$.

   However, in order to maintain sufficient accuracy one must be sure that the value of $T_1$ is not close to zero. Indeed, if any of the quantities

   $$T_1 = \det \begin{bmatrix} z_{i,i} & z_{i,n-i+1} \\ z_{n-i+1,i} & z_{n-i+1,n-i+1} \end{bmatrix} = 0$$

   (VII.2.3.1)

   then the method will break down which in this case column $i$ can be interchanged with any of the columns $i+1,i+2,...,n-i+1$ to obtain a non-zero value of the determinants (VII.2.3.1). If no such non-zero determinantal value exists, then $A$ is singular.
2. Alternatively, the Gauss-elimination method can be used in order to solve each system and therefore maintain accuracy by interchanging the equations when required. In this case we have the following algorithm:

2.a. Compute \( \rho = z_{i,i}^{-1} z_{i,n-i+1} \).

If \( \rho > 0 \) then the values of \( w_{j,i} \) and \( w_{j,n-i+1} \) are obtained by the following procedure:

2.b.1 \( m = z_{i,i}^{-1} z_{i,n-i+1} \)

2.c.1 \( w_{j,n-i+1} = (a_{j,i} z_{i,i} - m a_{j,i}) / z_{n-i+1,i} \) and

2.d.1 \( w_{j,i} = (a_{j,i} z_{i,i} - m a_{j,i}) / z_{i,i} \)

whereas if \( \rho < 0 \) then we have

2.b.2 \( m = z_{i,i}^{-1} z_{i,n-i+1} \)

2.c.2 \( w_{j,n-i+1} = (a_{j,i} z_{i,i} - m a_{j,i}) / z_{n-i+1,i} \) and

2.d.2 \( w_{j,i} = (a_{j,i} z_{i,i} - m a_{j,i}) / z_{i,i} \)

Under the assumptions made in Chapter I, it can be seen that the first method requires 3 time steps, one multiplication for evaluating \( T_1, T_2 \) and \( T_3 \) using 6 processors and one subtraction also for evaluating \( T_1, T_2 \) and \( T_3 \) with 3 processors and one division for evaluating \( w_{j,i} \) and \( w_{j,n-i+1} \) with 2 processors and because \((n-2i)\) such \((2\times2)\) systems have to be solved at each stage, a total number of \( 6(n-2i) \) processors are required for the solution of all the systems at stage \( i \). However, if the second method is used then the process requires 8 time steps (i.e., 3 subtractions, 2 multiplications and 3 divisions) with 2 processors and hence \( 2(n-2i) \) number of processors are required to solve all such systems at stage \( i \).

Moreover, the execution of the replacement operation (VII2.2.1) requires a negligible amount of time and a total number of \( n-2(i-1) \) processors. The modifications of the \( a_{i,j} \)'s in (VII.2.2.6) requires 3 time
steps, one multiplication with \(2(n-2)^2\) processors and 2 subtractions with \((n-2)^2\) processors. This modification can alternatively be achieved in 4 time steps, 2 multiplications with \((n-2)^2\) processors and 2 subtractions with \((n-2)^2\) processors.

Therefore, for the complete factorisation process, if Cramer's rule is used for the solution of the \((2*2)\) linear systems in (VII.2.2) then we shall require either a total number of \(6\left\lfloor \frac{n-1}{2} \right\rfloor\) time steps and a maximum number of \(P\) processors where:

\[
P = \max\{6(n-2), 2(n-2)^2\} \tag{VII.2.3.2}
\]

or a total number of \(7\left\lfloor \frac{n-1}{2} \right\rfloor\) time steps and a maximum number of \(P\) processors where:

\[
P = \max\{6(n-2), (n-2)^2\} \tag{VII.2.3.3}
\]

and if the Gauss-elimination method is employed then we shall need either a total of \(11\left\lfloor \frac{n-1}{2} \right\rfloor\) time steps and a maximum of \(2(n-2)^2\) number; of processors, or a total number of \(12\left\lfloor \frac{n-1}{2} \right\rfloor\) time steps and a maximum number of \(P\) processors where:

\[
P = \max\{2(n-2), (n-2)^2\} \tag{VII.2.3.4}
\]

From the above discussion the following table which illustrates the alternative costs of the factorisation process can be derived:

<table>
<thead>
<tr>
<th>Solution Method of the ((2*2)) Linear Systems</th>
<th>Total Number of Time Steps</th>
<th>Maximum Number of Processors Working in Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cramer's Rule</td>
<td>(6\left\lfloor \frac{n-1}{2} \right\rfloor)</td>
<td>(\max{6(n-2), 2(n-2)^2})</td>
</tr>
<tr>
<td></td>
<td>(7\left\lfloor \frac{n-1}{2} \right\rfloor)</td>
<td>(\max{6(n-2), (n-2)^2})</td>
</tr>
<tr>
<td>Gauss-elimination method</td>
<td>(11\left\lfloor \frac{n-1}{2} \right\rfloor)</td>
<td>(2(n-2)^2)</td>
</tr>
<tr>
<td></td>
<td>(12\left\lfloor \frac{n-1}{2} \right\rfloor)</td>
<td>(\max{2(n-2), (n-2)^2})</td>
</tr>
</tbody>
</table>

The cost of the factorisation process of the Q.I.F. method

**TABLE (VII.2.3.1)**
VII.3 SOLUTION OF THE LINEAR SYSTEM

From (VII.2.1.1) and (VII.2.1.2) it can be seen that the system

\[(WZ)x = b\]  \hspace{1cm} (VII.3.1)

can be solved instead of (VII.2.1.1), where we need to solve 2 related and simpler linear systems of the form:

\[Wy = b\]  \hspace{1cm} (VII.3.2)
\[Zx = y\]  \hspace{1cm} (VII.3.3)

The system (VII.3.2) is first solved for the intermediate vector \(y\) and then the final solution of the system (VII.2.1.1) can be obtained by solving the linear system (VII.3.3) for \(x\).

The solution of (VII.3.2) is obtained in \([n-1]\) steps where from the structure of the matrix \(W\) it can be seen that \(y_1\) and \(y_n\) are evaluated first, then \(y_2, y_{n-1}\) and so on working in pairs from the top and rear of the vector \(y\). In general and at the \(i^{th}\) stage we have:

\[
\begin{cases}
  y_i = b_i \\
y_{n-i+1} = b_{n-i+1}
\end{cases}
\]  \hspace{1cm} (VII.3.4)

and

\[
b_j = b_j - w_j i y_i - w_j y_{n-i+1}, \quad j = i+1(1)n-i
\]

and then we proceed to the next step.

Under the same assumption as before, in Chapter I, it can be seen that the complete evaluation of the components of the intermediate vector \(y\) requires a total number of \(3[n-1]\) time steps and a maximum number of \(2(n-2)\) processors.

For the solution of the linear system (VII.3.3), we proceed as follows:

If \(n\) is odd, we can find that

\[x_t = y_t / z_{t,t}, t = n+1 \quad \text{and} \quad j = i+1(1)n\]

and in preparation for the next step we have:

\[y_j = y_j - x_t z_{j,t} \quad \text{and} \quad j = i+1(1)n\]  \hspace{1cm} (VII.3.5)
The rest of the elements of the vector $x$ are found in pairs by solving \( \frac{n-1}{2} \) number of (2×2) linear systems in \( \frac{n-1}{2} \) distinct stages. In general, and at the \( i \)th stage we solve the (2×2) linear system:

\[
\begin{aligned}
& z_i, i^2x_i + 2z_i, n_i+1^2x_i+1 = y_i, i = \frac{n-1}{2}(-1)^l \\
& z_{n-i+1, i^2x_i+2z_i, n-i+1, n_i+1^2x_i+1 = y_{n-i+1}}
\end{aligned}
\]

(VII.3.7)

to compute \( x_i \) and \( x_{n-i+1} \). We then set

\[
y_j = y_j - x_j z_i, x_i z_j, \quad i = \frac{n-1}{2}(-1)^l \\
y_{j} = \begin{cases} 
  y_{j}, & j = 1(1) \text{i-1 & n-i(l)h} 
\end{cases}
\]  

(VII.3.8)

and proceed to the next stage.

However, if \( n \) is even, then all the components of vector \( x \) are found in pairs. To find all the pairs, the linear systems in (VII3.7) and the formula (VII3.8) are executed respectively for \( i = \left[ \frac{n}{2} \right](-1)^l \).

It can easily be seen that for the solution of the linear system (VII3.3), if Cramer's rule is used for solving the (2×2) linear systems in (VII3.7) then a total number of \( 6\left[ \frac{n-1}{2} \right] + \frac{3}{2} \cdot (3+(-1)^n) \) time steps and a maximum number of \( 6(n-2) \) processors are required while if the Gauss-elimination method is employed then a total number of \( 11\left[ \frac{n-1}{2} \right] + 7 \cdot 4(-1)^n \) time steps and a maximum number of \( 2(n-2) \) processors are needed.

Table (VII3.1) illustrates the alternative computational costs for the complete quadrant interlocking factorisation of the coefficient matrix \( A \) and the solution of the linear system (VII2.1.1).
<table>
<thead>
<tr>
<th>Solution Method of the (2x2) Linear Systems</th>
<th>Total Number of Time Steps</th>
<th>Maximum Number of Processors Working in Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cramer's Rule</td>
<td>[15\left(\frac{n-1}{2}\right) + \frac{3}{2}[3(-1)^n]]</td>
<td>max(6(n-2), 2(n-2)^2)</td>
</tr>
<tr>
<td></td>
<td>[16\left(\frac{n-1}{2}\right) + \frac{3}{2}[3(-1)^n]]</td>
<td>max(6(n-2), (n-2)^2)</td>
</tr>
<tr>
<td>Gauss-elimination method</td>
<td>[25\left(\frac{n-1}{2}\right) + 7(-1)^n]</td>
<td>2(n-2)^2</td>
</tr>
<tr>
<td></td>
<td>[26\left(\frac{n-1}{2}\right) + 7(-1)^n]</td>
<td>max(2(n-2), (n-2)^2)</td>
</tr>
</tbody>
</table>

Alternative computational costs for the complete quadrant interlocking factorisation of the coefficient matrix $A$ and the solution of the linear system

TABLE (VII.3.1)
Because of the rounding errors which occur during the quadrant interlocking factorisation of a coefficient matrix A, it can be said that the computed quadrant factors of A, i.e. W and Z, correspond to the exact factorisation of A+E where E is a perturbation matrix. In this section we attempt to obtain upper bounds for the matrix E. Our approach in doing so is similar to that of WILKINSON [42] for the error analysis of the Gaussian elimination method.

Before going into any detail of the process, we assume that:

1. The elements of the matrices W and Z are stored on their corresponding positions in matrix A as they are evaluated.
2. The \([\frac{n-1}{2}]\) steps of the factorisation process, as described in section (VII.2.2) are defined as a sequence of matrices, \(A^{(1)}, A^{(2)}, \ldots, A^{([\frac{n-1}{2}])}\)

where \(A^{(i)}\) denotes the complete evaluation of the elements which lie on the first and last i rows and columns of A and for simplicity-\(A^{([\frac{n-1}{2}])}\) denotes the evaluation of the centre element \(a_{\frac{n+1}{2}, \frac{n+1}{2}}\) of A when n is odd and the evaluation of the centre elements \(a_{\frac{n}{2}, \frac{n}{2}}, a_{\frac{n-1}{2}, \frac{n+1}{2}}, a_{\frac{n+1}{2}, \frac{n}{2}}\) when n is even.

Under the above assumptions, it can be seen that the elements \(a_{i,j}\) of A which will finally form the matrix Z, i.e.,

\[
\begin{align*}
& a_{i,j}, \text{for } \begin{cases} \quad i=1,2, \ldots, \left[\frac{n+1}{2}\right] \\
\qquad j=i,i+1, \ldots, n-i+1
\end{cases} \\
& a_{i,j}, \text{for } \begin{cases} \quad i=\left[\frac{n+1}{2}\right]+1, \ldots, n \\
\qquad j=(n-i+1), \ldots, i
\end{cases}
\end{align*}
\]

are modified until \(A^{(i)}\) has been obtained and then they remain constant.

If we characterise an error term, \(e_{i,j}^{(k)}\), representing the total rounding error which we may commit on modifying the elements \(a_{i,j}\) of A as defined in (VII.4.1) at step \(k\), \(k=1,2, \ldots, \left[\frac{n-1}{2}\right]\) of the elimination process, we then have:
\[ a_{i,j}^{(2)} = a_{i,j}^{(1)} - w_{i,j}^{(2)} a_{i,n}^{(1)} a_{i,j}^{(1)} + e_{i,j}^{(2)} \]
\[ a_{i,j}^{(3)} = a_{i,j}^{(2)} - w_{i,j}^{(2)} a_{i,n}^{(2)} a_{i,j}^{(2)} + e_{i,j}^{(3)} \]
\[ \vdots \]
\[ a_{i,j}^{(i)} = a_{i,j}^{(i-1)} - w_{i,j}^{(i-1)} a_{i,n}^{(i-1)} a_{i,j}^{(i-1)} + e_{i,j}^{(i)} \]

Summing the equations above we have:
\[ a_{i,j}^{(i)} = a_{i,j}^{(i-1)} - w_{i,j}^{(i-1)} a_{i,n}^{(i-1)} a_{i,j}^{(i-1)} + e_{i,j}^{(i)} \]

where:
\[ e_{i,j} = e_{i,j}^{(2)} + e_{i,j}^{(3)} + \cdots + e_{i,j}^{(i)} \]

Note that a similar expression can be obtained for the elements in the \((n-i+1)\)th row of \(A\), i.e.,
\[ e_{n-i+1,j} = e_{n-i+1,j}^{(2)} + e_{n-i+1,j}^{(3)} + \cdots + e_{n-i+1,j}^{(i)} \]

It is now evident that the total error, \(e_{i,j}\), consists of \((i-1)\) individual error terms occurring at each step of the elimination process for all \(a_{i,j}\) as defined in (VII.4.1).

The elements \(a_{i,j}\) of \(A\) which will finally form the matrix \(W\), i.e.,
\[ a_{i,j} \text{ for } j=1,2,\ldots,\left\lfloor \frac{n-1}{2} \right\rfloor \]
are modified until \(A^{(j)}\) is obtained. The elements \(a_{i,j}^{(j)}\) and \(a_{i,n-j+1}^{(j)}\), \(i=2j,2j+1,\ldots,n-2j+1\), are then used to evaluate the element \(w_{i,j}\) and \(w_{i,n-j+1}, i=2j,2j+1,\ldots,n-2j+1\). In general the error term for each of such elements is a summation of two types of errors:

a) An error term which is created as a result of the modification process and

b) an error term which is generated when evaluating the multipliers (i.e., the elements of matrix \(W\)).
In a similar process to (VII.4.2), we can show that the error term associated with the elements $a_{i,j}$ as defined in (VII.4.6) has $(j-1)$ error terms of the type indicated in a), i.e.,

$$a_{i,j}^{(2)} = a_{i,j}^{(1)} - w_{i,j} l_{i,j}^{(1)} - w_{i,j} n_{i,j}^{(1)}$$

$$a_{i,j}^{(3)} = a_{i,j}^{(2)} - w_{i,j} l_{i,j}^{(2)} - w_{i,j} n_{i,j}^{(2)}$$

$$a_{i,j}^{(j)} = a_{i,j}^{(j-1)} - w_{i,j} l_{i,j}^{(j-1)} - w_{i,j} n_{i,j}^{(j-1)}$$

and summing the above equations we have:

$$a_{i,j}^{(j)} = a_{i,j}^{(1)} - w_{i,j} l_{i,j}^{(1)} - w_{i,j} n_{i,j}^{(1)} - \ldots - w_{i,j} l_{i,j}^{(j-1)} - w_{i,j} n_{i,j}^{(j-1)} + e_{i,j}^{(2)} + \ldots + e_{i,j}^{(j)}$$

where

$$e_{i,j} = e_{i,j}^{(2)} + \ldots + e_{i,j}^{(j)}$$

Moreover, in evaluating the corresponding multipliers we are expected to increase the above summation (VII.4.8) by another error term ($e_{i,j}^{(j+1)}$) and therefore we can conclude that the total error ($e_{i,j}$) corresponding to the elements $a_{i,j}$ as defined in (VII.4.6) is equal to the summation of $j$ error terms, i.e., $j-1$ terms of the type in a), and one term of the type in b) and,

$$e_{i,j} < j^* \zeta$$

where $\zeta$ is the maximum in absolute value of these two types of error.

From the above discussions which, in fact, describes the structure of the error terms associated with each element of the matrices $Z$ and $W$, we can derive the following two matrices which illustrates the number of error terms for each $a_{i,j}$ of $A$. 
a) For \( n \) odd

\[
N_1 = \begin{bmatrix}
0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
1 & 1 & 1 & \ldots & 1 & 1 & 1 \\
1 & 2 & 2 & \ldots & 2 & 2 & 1 \\
1 & 2 & 3 & \ldots & 3 & 2 & 1 \\
\vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \vdots \\
1 & 2 & 3 & \ldots & 3 & 2 & 1 \\
1 & 2 & 2 & \ldots & 2 & 2 & 1 \\
1 & 1 & 1 & \ldots & 1 & 1 & 1 \\
0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[ (VII.4.10) \]

b) For \( n \) even

\[
N_2 = \begin{bmatrix}
0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
1 & 1 & 1 & \ldots & 1 & 1 & 1 \\
1 & 2 & 2 & \ldots & 2 & 2 & 1 \\
1 & 2 & 3 & \ldots & 3 & 2 & 1 \\
\vdots & \vdots & \ddots & \ddots & \vdots & \vdots & \vdots \\
1 & 2 & 3 & \ldots & 3 & 2 & 1 \\
1 & 2 & 2 & \ldots & 2 & 2 & 1 \\
1 & 1 & 1 & \ldots & 1 & 1 & 1 \\
0 & 0 & 0 & \ldots & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[ (VII.4.11) \]

We now attempt to find the maximum value for each type of error described above.

i) The Reduction Error

The formula (VII.2.2.6) in floating-point notation can be written as:

\[
a_{i,j}^{(k+1)} = f_k[a_{i,j}^{(k)} - w_i, a_{i,k}^{(k)} - w_i, a_{i,k+1}^{(k)}]
\]

\[ (VII.4.12) \]

\[
\approx (a_{i,j}^{(k)} - w_i, a_{i,k}^{(k)}(1 + \varepsilon_1) - w_i, a_{i,k+1}^{(k)}(1 + \varepsilon_2))(1 + \varepsilon_3)
\]

where \[ |\varepsilon_i| < 2^{-t} \]

Hence,
\[
\frac{a_{i,j}^{(k+1)}}{1+\epsilon_3} = a_{i,j}^{(k)} - \sum_{k=\min(i,j)}^{\max(i,j)} \sum_{i=1}^{n-k+1} a_{i,k,j}^{(k)} w_{i,k}^{(k)} w_{i,n-k+1,j}^{(k)} 
\]

(VII.4.14)

Therefore, the difference between the actual \(a_{i,j}^{(k+1)}\) and the computed \(a_{i,j}^{(k+1)}\) is as follows:

\[
\epsilon_{i,j}^{(k+1)} = a_{i,j}^{(k+1)} - a_{i,j}^{(k)} = \frac{a_{i,j}^{(k+1)} - a_{i,j}^{(k)}}{1+\epsilon_3} \leq \frac{1}{1+\epsilon_3} \quad \text{(VII.4.15)}
\]

If we assume that we have applied some form of pivoting such that all the elements of the matrix \(W\) are less than or equal to one and that \(|a_{i,j}^{(r)}| \leq g\) for all values of \(r\), then we can conclude from (VII.4.15) and (VII.4.13) that

\[
|\epsilon_{i,j}^{(k+1)}| < \frac{g^2}{1-2^{-t}} + \frac{g^2}{1+2^{-t}} \leq \frac{g^2}{1-2^{-t}} \quad \text{(VII.4.16)}
\]

Assuming that

\[
\frac{1}{1-2^{-t}} \leq 1.01 \quad \text{(say)} \quad \text{(VII.4.17)}
\]

we have the result

\[
|\epsilon_{i,j}^{(k+1)}| \leq 3.01 g^2 2^{-t} \quad \text{(VII.4.18)}
\]

ii) The Error for the Evaluation of the Components of the Matrix \(W\)

For the determination of an upper bound for the error committed when evaluating the elements of the matrix \(W\), we assume that the set of \((2*2)\) linear systems given by (VII.2.2,5) are solved according to Cramer's rule as described in (VII.2.2). Therefore, we have,

\[
\begin{align*}
\mathbf{w}_{i,j} &= \mathbf{x} \left[ \left( a_{i,j}^{(r)} + a_{i,p}^{(r)} + a_{j,p}^{(r)} \right) / a_{i,p}^{(r)} a_{j,p}^{(r)} \right] \\
&= \left\{ \left( a_{i,j}^{(r)} - a_{i,j}^{(r)} a_{p,j}^{(r)} a_{j,p}^{(r)} \right) / a_{i,p}^{(r)} a_{j,p}^{(r)} \right\} + \left( a_{i,j}^{(r)} - a_{i,j}^{(r)} a_{p,j}^{(r)} a_{j,p}^{(r)} \right)
\end{align*}
\]

(VII.4.19)

where \(p = n-j+1\).

Hence we have:

\[
\begin{align*}
\mathbf{w}_{i,j} &= \frac{\left\{ \left( a_{i,j}^{(r)} + a_{i,p}^{(r)} + a_{j,p}^{(r)} \right) / a_{i,p}^{(r)} a_{j,p}^{(r)} \right\} + \left( a_{i,j}^{(r)} - a_{i,j}^{(r)} a_{p,j}^{(r)} a_{j,p}^{(r)} \right)}{a_{i,p}^{(r)} a_{j,p}^{(r)} a_{p,j}^{(r)} a_{j,p}^{(r)}} \\
&= \frac{\left( 1+\epsilon_{i,j} \right) \left( 1+\epsilon_{p,j} \right) \left( 1+\epsilon_{j,p} \right) \left( 1+\epsilon_{i,p} \right)}{\left( 1+\epsilon_{i} \right) \left( 1+\epsilon_{j} \right) \left( 1+\epsilon_{p} \right) \left( 1+\epsilon_{j} \right)} \\
&= \frac{\left( 1+\epsilon_{i} \right) \left( 1+\epsilon_{j} \right) \left( 1+\epsilon_{p} \right) \left( 1+\epsilon_{j} \right)}{\left( 1+\epsilon_{i} \right) \left( 1+\epsilon_{j} \right) \left( 1+\epsilon_{p} \right) \left( 1+\epsilon_{j} \right)}
\end{align*}
\]

(VII.4.19)

where

\[
|\epsilon_i| < 2^{-t} \quad i=1,2,\ldots,7. \quad \text{(VII.4.20)}
\]
The expression (VII.4.19) can be rewritten as:

\[
\frac{w_{i,j}(1+\epsilon_6)}{(1+\epsilon_7)(1+\epsilon_6)} = \frac{a_{i,p}a_{p,j} - a_{i,j}a_{j,p}}{a_{p,j}a_{j,p} - a_{i,j}a_{j,p}} + \frac{\epsilon_{1,a_{i,p}a_{p,j} - a_{i,j}a_{j,p}}}{a_{p,j}a_{j,p} - a_{i,j}a_{j,p}}\]

(VII.4.21)

and therefore:

\[
\frac{a_{i,p}a_{p,j} - a_{i,j}a_{j,p}}{a_{p,j}a_{j,p} - a_{i,j}a_{j,p}} = \frac{w_{i,j}(1+\epsilon_6)}{(1+\epsilon_7)(1+\epsilon_6)} - \frac{w_{i,j}(1+\epsilon_7)}{(1+\epsilon_6)(1+\epsilon_7)} + \frac{\epsilon_{3,a_{i,p}a_{p,j} - a_{i,j}a_{j,p}}}{a_{p,j}a_{j,p} - a_{i,j}a_{j,p}}\]

(VII.4.22)

The difference between the actual \(w_{i,j}\) and the computed \(w_{i,j}\) can now be evaluated

\[
\epsilon_{i,j}^{(j+1)} = \frac{a_{i,p}a_{p,j} - a_{i,j}a_{j,p}}{a_{p,j}a_{j,p} - a_{i,j}a_{j,p}} \rightline{w_{i,j}(1+\epsilon_6)} - \frac{w_{i,j}(1+\epsilon_7)}{(1+\epsilon_6)(1+\epsilon_7)} + \frac{\epsilon_{3,a_{i,p}a_{p,j} - a_{i,j}a_{j,p}}}{a_{p,j}a_{j,p} - a_{i,j}a_{j,p}}\]

(VII.4.23)

Ignoring any \(\epsilon_6^*\epsilon_7\) we have:

\[
\epsilon_{i,j}^{j+1} = \frac{w_{i,j}(\epsilon_7^*\epsilon_5 - \epsilon_6^*\epsilon_7)}{\epsilon_5^*\epsilon_7} - \frac{w_{i,j}(1+\epsilon_7)}{(1+\epsilon_6)(1+\epsilon_7)} + \frac{\epsilon_{3,a_{i,p}a_{p,j} - a_{i,j}a_{j,p}}}{a_{p,j}a_{j,p} - a_{i,j}a_{j,p}}\]

(VII.4.24)

Suppose that \(\alpha\) is the minimum of all the denominators in (VII.4.23) i.e.,

\[
\alpha = \text{Min } \left| \begin{array}{cc} a_{i,i} & a_{i,p} \\ a_{p,i} & a_{p,p} \end{array} \right|, i=1,2,...,\left(\frac{n-1}{2}\right)\]

(VII.4.24)

then we have,
\[
\begin{align*}
\frac{\epsilon^a_{5,j} p^a_{j,p} - \epsilon^a_{j} p^a_{p,j}}{a_p j^a_{p,j} p^a_{p,j}} < \frac{\epsilon^2}{a}(|\epsilon_3| + |\epsilon_4|), \\
\text{(VII. 4.25)}
\end{align*}
\]

and
\[
\begin{align*}
\frac{\epsilon^a_{l,j} p^a_{p,j} - \epsilon^a_{2,i} j^a_{i,p}}{a_p j^a_{j,p} p^a_{j,p}} < \frac{\epsilon^2}{a}(|\epsilon_1| + |\epsilon_2|). \\
\text{(VII. 4.26)}
\end{align*}
\]

Hence, assuming that:
\[
\frac{1}{1 - 2^{-t+1}} < 1.03 \text{ (say) (VII. 4.27)}
\]

and
\[
\frac{2^{-t}}{1 - 2^{-t+1}} < 0.03 \text{ (say) (VII. 4.28)}
\]

we have
\[
|\epsilon_{i,j}^{(j+1)}| < 2^{-t} \left(3.09 \cdot \frac{4.12}{a} g^2\right). \\
\text{(VII. 4.28)}
\]

From all we have discussed in this section we can conclude that the Q.I.F. of a coefficient matrix \(A\) is true for computed values of the matrices \(W\) and \(Z\) such that:
\[
W^*Z = A + E \quad \text{(VII. 4.29)}
\]

where \(E\) is given by
\[
\begin{align*}
\begin{cases}
|E| < \rho N_1, & \text{for } n \text{ odd} \\
|E| < \rho N_2, & \text{for } n \text{ even}
\end{cases} \\
\text{(VII. 4.30)}
\end{align*}
\]

where,
\[
\rho = \max\{2^{-t} (3.09 \cdot \frac{4.12}{a} g^2), g^{2-t} \cdot 3.01\} \quad \text{(VII. 4.31)}
\]

and \(N_1\) and \(N_2\) are the matrices given by (VII A.10) and (VII A.11) respectively.
VII.5 THE ERROR ANALYSIS OF THE SOLUTION PROCESS

VII.5.1 The Error Analysis of the Linear System $Wy=b$

The algorithm presented in section (VII.3) for the solution of the linear system $\Omega \tilde{w}_2$, illustrates that the first and last components of the vector $y$ can be evaluated as:

$$\begin{cases} y_1 = b_1 \\ y_n = b_n \end{cases} \quad (VII.5.1.1)$$

Then, in floating-point arithmetic the evaluation of the elements $y_2$ and $y_{n-1}$ of vector $y$ can be expressed as:

$$\begin{cases} y_2 = \left( b_2 - w_2, 1 \right) \left( 1 + \varepsilon_2, 1 \right) v_1(1+\varepsilon_1) - w_2, n \cdot y_n(1+\varepsilon_2,n) \right)(1+\varepsilon_2) \\ y_{n-1} = \left( b_{n-1} - w_{n-1}, 1 \right) \left( 1 + \varepsilon_{n-1}, 1 \right) v_1(1+\varepsilon_{n-1},1) - w_{n-1}, n \cdot y_n(1+\varepsilon_{n-1},n) \right)(1+\varepsilon_{n-1}) \end{cases} \quad (VII.5.1.2)$$

From the above equations, further analysis demonstrates that the structure of the solution process for the elements $y_2$ and $y_{n-1}$ of the vector $y$ can be written as:

$$\begin{cases} y_2 = b_2(1+\varepsilon_1)(1+\varepsilon_2) - w_2, 1 \cdot y_1(1+\varepsilon_1) - w_2, n \cdot y_n(1+\varepsilon_2,n) \right)(1+\varepsilon_2) \\ y_{n-1} = b_{n-1}(1+\varepsilon_{n-1})(1+\varepsilon_2) - w_{n-1}, 1 \cdot y_1(1+\varepsilon_{n-1},1) - w_{n-1}, n \cdot y_n \right)(1+\varepsilon_{n-1}) \end{cases} \quad (VII.5.1.3)$$

where

$$|\varepsilon_1| < 2^{-t} \quad , \quad |\varepsilon_{n-1}| < 2^{-t} \quad (VII.5.1.4)$$

and

$$|\varepsilon_{i,j}| < 2^{-t} \quad .$$

Finally, after further analysis, we have the solution for $y_r$ expressed in the form:

$$y_r = b_r(1+E_{r,r}) - \sum_{i=1}^{r-1} \left[ w_{r,i} y_1(1+\varepsilon_{r,i}) - w_{r,n-i+1} y_{n-i+1}(1+\varepsilon_{r,n-i+1}) \right]$$

for $r \equiv 2(1)\left[ \frac{n+1}{2} \right] \quad (VII.5.1.5)$

where:

$$\begin{cases} 1+E_{r,1} = (1+\varepsilon_{r,1})(1+\varepsilon_1)(1+\varepsilon_2) \ldots (1+\varepsilon_{2r-2}) \\ 1+E_{r,n} = (1+\varepsilon_{r,n})(1+\varepsilon_2)(1+\varepsilon_3) \ldots (1+\varepsilon_{2r-2}) \end{cases}$$
\[
\begin{align*}
1 + \varepsilon_{r,2} &= (1 + \varepsilon_{r,2}) (1 + \varepsilon_3) (1 + \varepsilon_4) \cdots (1 + \varepsilon_{2r-2}) \\
1 + \varepsilon_{r,n-1} &= (1 + \varepsilon_{r,n-1}) (1 + \varepsilon_4) (1 + \varepsilon_5) \cdots (1 + \varepsilon_{2r-2}) \\
\vdots
\end{align*}
\]
(VII.5.1.6)

and
\[
\begin{align*}
1 + \varepsilon_{r,r-1} &= (1 + \varepsilon_{r,r-1}) (1 + \varepsilon_{2r-3}) (1 + \varepsilon_{2r-2}) \\
1 + \varepsilon_{r,n-r+2} &= (1 + \varepsilon_{r,n-r+2}) (1 + \varepsilon_{2r-2})
\end{align*}
\]

where
\[
|\varepsilon_{r,i}| < 2^{-t}, \quad i = 1, 2, \ldots, 2r-2
\]
(VII.5.1.7)

\[
|\varepsilon_{r,j}| < 2^{-t}, \quad j = 1(1)r-1 \& n(-1)n-r+2.
\]
(VII.5.1.8)

From (VII.5.1.6) we have
\[
\begin{align*}
1 + \varepsilon_{r,i} &= (1 + \varepsilon_{r,i}) \frac{2r-2}{j=2i-1} (1 + \varepsilon_1) \\
1 + \varepsilon_{r,n-i+1} &= (1 + \varepsilon_{r,n-i+1}) \frac{2r-2}{j=2i} (1 + \varepsilon_1)
\end{align*}
\]
(VII.5.1.9)

Hence, from the above expressions in (VII.5.1.9) and (VII.5.1.7), (VII.5.1.8) we have:
\[
\begin{align*}
(1-2^{-t})^{2r-2i+1} + \varepsilon_{r,i} &< (1+2^{-t})^{2r-2i+1} \\
(1-2^{-t})^{2r-2i} + \varepsilon_{r,n-i+1} &< (1+2^{-t})^{2r-2i} \\
(1-2^{-t})^{2r-2} + \varepsilon_{r,r} &< (1+2^{-t})^{2r-2}
\end{align*}
\]
(VII.5.1.10)

It can be seen from WILKINSON [43] that if,
\[
(1-2^{-t})^p < 1 + \varepsilon < (1+2^{-t})^p
\]
(VII.5.1.11)

then
\[
|\varepsilon| < (1+2^{-t})^{t_1}
\]
(VII.5.1.12)

where
\[
t_1 = t - \log_2(1.06) = t - 0.08406.
\]
(VII.5.1.13)

Therefore, from the inequalities given by (VII.5.1.10) we can conclude that:
\[
\begin{cases}
|E_{r,i}| < (2r-2i+1)2^{-t_1} \\
|E_{r,n-i+1}| < (2r-2i)2^{-t_1} \\
|E_{r,r}| < (2r-2)2^{-t_1}
\end{cases}
\quad (\text{VII.5.1.14})
\]

The formula (VII.5.1.5) can now be rewritten as:

\[
w_{r,1}y_1(1+E_{r,1}) + w_{r,n}y_n(1+E_{r,n}) + \ldots + w_{r,r-1}y_{r-1}(1+E_{r,r-1}) + w_{r,n-r+2}y_{n-r+2} + (1+E_{r,n-r+2})y_r = b_r(1+E_{r,r})
\]

for \( r = 2(1)^{\frac{n+1}{2}} \)

(VII.5.1.15)

and the above expression illustrates that the computed vector \( \gamma \) is the exact solution of the linear system:

\[
(W + \delta W)\gamma = b + \delta b
\]

(VII.5.1.16)

where \( \delta W \) and \( \delta b \) for a (7x7) matrix is illustrated as follows:

\[
\delta b \leq 2^{-t_1}
\]

(VII.5.1.17)

and

\[
\delta W \leq 2^{-t_1}
\]

(VII.5.1.18)
VII.5.2 The Error Analysis of the Linear System \( Z \times y \)

We saw, in section (VII.3) that the complete solution of the linear system (VII.3.3) is obtained by solving the number \( \left[ \frac{n}{2} \right] \) of \((2 \times 2)\) linear systems of the form:

\[
\begin{cases}
z_{r,r} x^r + z_{r,n-r+1} x^{n-r+1} = y^*_r \\
z_{n-r+1,r} x^r + z_{n-r+1,n-r+1} x^{n-r+1} = y^*_{n-r+1}
\end{cases}
\] (VII.5.2.1)

where \( y^*_r \) and \( y^*_{n-r+1} \) are the modified values of \( y_r \) and \( y_{n-r+1} \) respectively according to the formula (VII3.8) which under the same analysis obtained from computing (VII.3.4.6) in the previous section, we can conclude that:

\[
y^*_r = y^*_r (1 + E_{r,r}) - \sum_{i=r+1}^{[n/2]} (z_{r,i} x_i (1 + E_{r,i}) - z_{r,n-i+1} x^{n-i+1}) (1 + E_{r,n-i+1})
\]

\[
y^*_{n-r+1} = y^*_{n-r+1} (1 + E_{n-r+1,n-r+1}) - \sum_{i=r+1}^{[n/2]} (z_{n-r+1,i} x_i (1 + E_{n-r+1,i}) - z_{n-r+1,n-r+1} x^{n-r+1}) (1 + E_{n-r+1,n-r+1})
\] (VII.5.2.2)

where:

\[
\begin{align*}
1 + E_{r,i} &= (1 + \epsilon_{r,i}) \prod_{j=n-2i+1}^{n-2r} (1 + \epsilon_j), \\
1 + E_{n-r+1,i} &= (1 + \epsilon_{n-r+1,i}) \prod_{j=n-2i+1}^{n-2r} (1 + \bar{\epsilon}_j), \\
1 + E_{r,n-i+1} &= (1 + \epsilon_{r,n-i+1}) \prod_{j=n-2i+2}^{n-2r} (1 + \epsilon_j), \\
1 + E_{n-r+1,n-i+1} &= (1 + \epsilon_{n-r+1,n-i+1}) \prod_{j=n-2i+2}^{n-2r} (1 + \bar{\epsilon}_j), \\
1 + E_{r,r} &= \prod_{j=1}^{n-2r} (1 + \epsilon_j), \\
1 + E_{n-r+1,n-r+1} &= \prod_{j=1}^{n-2r} (1 + \bar{\epsilon}_j), \quad \text{for } i = r+1(1) \left[ \frac{n}{2} \right]
\end{align*}
\] (VII.5.2.3)

with

\[ |\epsilon_j| < 2^{-t}, \quad |\bar{\epsilon}_j| < 2^{-t} \] (VII.5.2.4)

and

\[ |\epsilon_{i,j}| < 2^{-t} \]
Now to solve the \((2*2)\) linear system

\[
\begin{align*}
\begin{cases}
 z_r, r, x_r, r, p, x_p = y_r^* \\
 z_p, r, x_r, r, p, x_p = y_p^*
\end{cases}, \quad p=n-r+1
\end{align*}
\]  

(VII.5.2.5)

for the unknowns \(x_r\) and \(x_p\) using Cramer's rule as before we have

\[
x_r = \frac{y_r^* p, r, p, (1+\varepsilon_3) - y_r^* z, r, p, p, (1+\varepsilon_4)}{z_r, z, p, r, p, (1+\varepsilon_1) - z_r, z, p, r, p, (1+\varepsilon_2)} \cdot \frac{(1+\varepsilon_5) (1+\varepsilon_6)}{(1+\varepsilon_7)}
\]

(VII.5.2.6)

and similarly for \(x_p\) we have:

\[
x_p = \frac{y_p^* z, p, r, (1+\varepsilon_8) - y_p^* z, p, r, (1+\varepsilon_9)}{z_r, z, p, r, p, (1+\varepsilon_1) - z_r, z, p, r, p, (1+\varepsilon_2)} \cdot \frac{(1+\varepsilon_10) (1+\varepsilon_11)}{(1+\varepsilon_12)}
\]

(VII.5.2.7)

The above two expressions can be rewritten as:

\[
x_r = \frac{y_r^* (1+\varepsilon_5) (1+\varepsilon_6) z, p, r, p, (1+\varepsilon_3) - y_r^* (1+\varepsilon_5) (1+\varepsilon_6) z, r, p, p, (1+\varepsilon_4)}{z_r, z, r, p, (1+\varepsilon_1) - z_r, z, p, p, (1+\varepsilon_2)}
\]

(VII.5.2.8)

and

\[
x_p = \frac{y_p^* (1+\varepsilon_10) (1+\varepsilon_11) z, p, r, (1+\varepsilon_8) - y_p^* (1+\varepsilon_10) (1+\varepsilon_11) z, r, r, (1+\varepsilon_9)}{z_r, z, r, p, (1+\varepsilon_1) - z_r, z, p, p, (1+\varepsilon_2)}
\]

(VII.5.2.9)

where 

\[|\varepsilon_i| < 2^{-c}, \quad i=1,2,\ldots,11.\]

In order to simplify what could be an extremely difficult analysis we assume that we can associate the new error terms to \(y_r^*\) and \(y_p^*\) from which we can easily see that the expressions (VII.5.2.8) and (VII.5.2.9) can be rewritten as:

\[
x_i = \frac{y_r^* z, r, r, x_r, r, p, p, (1+\varepsilon_3) - y_r^* z, r, p, p, (1+\varepsilon_4)}{z_r, z, r, p, (1+\varepsilon_1) - z_r, z, p, p, (1+\varepsilon_2)}
\]

(VII.5.2.10)

and

\[
x_p = \frac{y_p^* z, p, r, r, x_r, r, (1+\varepsilon_8) - y_p^* z, r, r, p, (1+\varepsilon_9)}{z_r, z, r, p, (1+\varepsilon_1) - z_r, z, p, p, (1+\varepsilon_2)}
\]

(VII.5.2.11)

where, from (VII.5.2.2) and (VII.5.2.3) we have:

\[
y_r^* = y_r (1+E_i)_{r, r} - \sum_{i=r+1}^{[n/2]} \left[ z, r, x, i, (1+E_i)_{r, r, i} - z, r, n-i+1, x, n-i+1, (1+E_i)_{r, r, n-i+1} \right]
\]

(VII.5.2.12)
Note that a similar formula can be obtained for $y_p$. From (VII.5.2.12) we can conclude that:

\[
\begin{align*}
|E_{r,i}^i|_{r} &< (2i-2r+3) \cdot 1, \quad i=r+1(1)[\frac{n}{2}]-1 \\
|E_{r,n-i+1}^i|_{r} &< (2i-2r+2) \cdot 1, \quad t_1=t-0.08406 \\
|E_{r,r}^i|_{r} &< (n-2r+2) \cdot 1
\end{align*}
\]

Therefore, it can be said that under certain assumptions the computed vector $x$ in the system (VII.3.3) is true for the following system

\[
(Z+\delta Z)\tilde{x} = y+\delta y
\]

where $\delta Z$ and $\delta y$ for a (6*6) matrix example are:

\[
\begin{bmatrix}
|z_{1,1}| & 5|z_{1,2}| & 7|z_{1,3}| & 6|z_{1,4}| & 4|z_{1,5}| & 1|z_{1,6}| \\
|z_{2,1}| & 5|z_{2,2}| & 7|z_{2,3}| & 6|z_{2,4}| & 4|z_{2,5}| & 1|z_{2,6}| \\
|z_{3,1}| & 5|z_{3,2}| & 7|z_{3,3}| & 6|z_{3,4}| & 4|z_{3,5}| & 1|z_{3,6}| \\
|z_{4,1}| & 5|z_{4,2}| & 7|z_{4,3}| & 6|z_{4,4}| & 4|z_{4,5}| & 1|z_{4,6}| \\
|z_{5,1}| & 5|z_{5,2}| & 7|z_{5,3}| & 6|z_{5,4}| & 4|z_{5,5}| & 1|z_{5,6}| \\
|z_{6,1}| & 5|z_{6,2}| & 7|z_{6,3}| & 6|z_{6,4}| & 4|z_{6,5}| & 1|z_{6,6}|
\end{bmatrix}
\]

and

\[
\begin{bmatrix}
6|y_1| \\
4|y_2| \\
2|y_3| \\
2|y_4| \\
4|y_5| \\
6|y_6|
\end{bmatrix}
\]
A MODIFICATION OF THE Q.I.F. METHOD

VIIL.1 General Description

In this section we describe the modified Q.I.F. method whereby the two matrix factors, W and Z are chosen to be slightly different in structure from the matrix factors involved in the original method. See EVANS & HADJIDIMOS [49].

This method factorises the coefficient matrix A into two matrices W and Z where

\[ A = WZ \]  \hspace{1cm} (VII.6.1.1)

with the matrices W and Z defined as follows: if \( W_i \) and \( Z_i \), \( i=1,2,\ldots,n \) are the column vectors of the matrices \( W \) and \( Z^T \), then we have:

\[ W = \begin{bmatrix} W_1 & W_2 & \cdots & W_n \end{bmatrix} \]  \hspace{1cm} (VII.6.1.2)

and

\[ Z^T = \begin{bmatrix} Z_1^T & Z_2^T & \cdots & Z_n^T \end{bmatrix} \]

where the column vectors \( W_i \) and \( Z_i \) are of the following general form:

\[ W_i = \begin{bmatrix} 0, \ldots, 0, 1, w_{i+1,i}, \ldots, w_{n-i+1,i}, 0, \ldots, 0 \end{bmatrix}^T, \ i=1(1)\left[ \frac{n}{2} \right] \]  \hspace{1cm} (VII.6.1.3)

\[ W_i = \begin{bmatrix} 0, \ldots, 0, 1, \ldots, 1, 0 \end{bmatrix}^T, \ i=\left[ \frac{n}{2} \right]+1 \]

and

\[ Z_i = \begin{bmatrix} 0, \ldots, 0, z_{1,i}, \ldots, z_{n-i+1,i}, 0, \ldots, 0 \end{bmatrix}^T, \ i=1(1)\left[ \frac{n+1}{2} \right] \]  \hspace{1cm} (VII.6.1.4)

\[ Z_i = \begin{bmatrix} 0, \ldots, 0, 0, \ldots, 0 \end{bmatrix}^T, \ i=\left[ \frac{n+1}{2} \right]+1(1)n \]

from which it can be easily seen that

\[ A = \sum_{i=1}^{n} W_i Z_i^T \]  \hspace{1cm} (VII.6.1.5)

For \( n=5 \), the matrices \( W \) and \( Z \) have the following forms:
Similarly for $n=6$ we have:

$$W = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & w_{2,6} \\
0 & w_{3,1} & w_{3,2} & 0 & 0 & w_{3,6} \\
0 & w_{4,1} & w_{4,2} & w_{4,3} & 1 & w_{4,6} \\
0 & w_{5,1} & w_{5,2} & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

and

$$Z = \begin{bmatrix}
z_{1,1} & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} & z_{1,6} \\
0 & z_{2,2} & z_{2,3} & z_{2,4} & z_{2,5} & 0 \\
0 & 0 & z_{3,3} & z_{3,4} & 0 & 0 \\
0 & 0 & 0 & z_{4,4} & 0 & 0 \\
0 & 0 & z_{5,3} & z_{5,4} & z_{5,5} & 0 \\
0 & z_{6,2} & z_{6,3} & z_{6,4} & z_{6,5} & z_{6,6}
\end{bmatrix}$$

**VII.6.2 The Computation of $W$ and $Z$ Matrices**

The elements of the matrices $W$ and $Z$ are evaluated in $[\frac{n+1}{2}]$ stages, where at the commencement of the $k^\text{th}$ stage we define the matrix $A_k$, $k=1(1)[\frac{n-1}{2}]$ as follows:

$$A_1 = A,$$

$$A_k = A - \sum_{i=1}^{k-1} W_i Z_i - \sum_{i=n-k+2}^{n} W_i Z_i, \quad k=2(1)[\frac{n-1}{2}]$$

It can easily be seen that the first and the last $(k-1)$ rows and
columns of any matrix \( A_k \) defined by (VII.6.2.1) are zero and to evaluate the elements which are different from 0 and 1 in the \( k^{th} \) and \((n-k+1)^{th}\) rows and columns of \( W \) and \( Z \) we have the following algorithm:

a) \( z_{k,j}^{(k)} = a_{k,j}^{(k)}, \quad j = k(1)n-k+1 \)

b) \( w_{j,k}^{(k)} = a_{j,k}^{(k)}/z_{k,k}^{(k)}, \quad j = k+1(1)n-k+1 \)  \( \text{(VII.6.2.1)} \)

c) \( z_{n-k+1,j}^{(k)} = a_{n-k+1,j}^{(k)}w_{n-k+1,k}z_{k,j}^{(k)}, \quad j = k+1(1)n-k+1 \)

d) \( w_{j,n-k+1}^{(k)} = (a_{j,n-k+1}^{(k)}w_{j,1}z_{k,n-k+1})/z_{n-k+1,n-k+1}, \quad j = k+1(1)n-k \)

and e) \( A_{k+1} = A_k W_k Z_k^T W_{n-k+1} z_{n-k+1}^T \)

for all \( k = 1(1)(n-1)/2 \). To complete the computation of all the elements of matrices \( W \) and \( Z \) a further step (VII.6.2.2a) has to be carried out for \( n \) odd, to evaluate the centre element \( z_{n+1/2, n+1/2} \) whilst if \( n \) is even the further steps (VII.6.2.2a) and (VII.6.2.2c) have to be executed in order to find the central elements \( z_{k,k}, z_{k,k+1}, w_{k+1,k} \) and \( z_{k+1,k+1} \), where \( k = n/2 \).

Under the same assumption as before, in Chapter I, it can be seen that the complete evaluation of the elements of the \( W \) and \( Z \) matrices requires either a total number of \( 9[n(n-1)/2] + 3/2(1+(-1)^n) \) time steps and a maximum number of \( 2(n-1)^2 \) processors or a total number of \( 10[n(n-1)/2] + 3/2(1+(-1)^n) \) time steps and a maximum number of \( (n-1)^2 \) processors.

### VII.6.3 Solution of the Linear System

For the solution of the linear system (VII.2.1.1) we need to solve 2 related and simpler linear systems of the form:

\[
Wx = b
\]

and

\[
Zx = \gamma
\]

where the linear system (VII.6.3.1) is first solved for the intermediate vector \( \gamma \) and then the final solution of the system (VII.2.1.1) can be
obtained by solving the linear system (VII.6.3.2) for vector $x$.

To solve system (VII.6.3.1) we let

$$W_x = b = b^{(1)}$$

so that

$$\sum_{i=1}^{n} y_i W_i = b^{(1)}.$$  \hspace{1cm} (VII.6.3.4)

Hence,

a) $y_k = b^{(k)}_k$

b) $y_{n-k+1} = b^{(k)}_{n-k+1} - y_k w_{n-k+1,k}$

c) $b^{(k+1)} = b^{(k)} - y_k w_{k} - y_{n-k+1} w_{n-k+1}$

for $k=1(1)[\frac{n-1}{2}]$. If $n$ is odd, step (VII.6.3.5a) has to be executed

for $k=[\frac{n+1}{2}]$ to find the centre element $y_k$, while for $n$ even, steps

(VII.6.3.5a) and (VII.6.3.5b) have to be executed for $k=[\frac{n+1}{2}]$ to find the
two elements in the middle $y_k$ and $y_{k+1}$. It can be easily seen that for
the solution of this system a total number of $5[\frac{n-1}{2}]+1+(-1)^n$ time steps
and a maximum number of $2(n-1)$ processors are required.

To solve the linear system (VII.6.3.2) we proceed as follows, first

we

$$Zx = y = y^{(1)}$$

so that

$$\sum_{i=1}^{n} x_i Z_i = y^{(1)}$$

where $Z_i$, $i=1,2,...,n$ are the column vectors of the matrix $Z$, i.e.,

$$Z = [Z_1 Z_2^* ... Z_n^*].$$

Hence if we set $i=[\frac{n+1}{2}]$ then for $n$ odd, we can find that:

$$\begin{cases}
  x_i = y_i^{(1)}/z_i^* & \text{if } i \neq i \\
  x_i = (y_i^{(1)}-x_i z_i^*)/z_i^* & \text{if } i = i
\end{cases}$$

and

while for $n$ even we have:

$$\begin{cases}
  x_{i+1} = y_{i+1}^{(1)}/z_{i+1}^{*1,i+1} \\
  x_i = (y_i^{(1)}-x_i z_i^{*1,i+1})/z_i^* 
\end{cases}$$
and

\[ y^{(2)} = y^{(1)} - x_z^* z_k^* x_{l+1} z_{l+1}^* \]  \hspace{1cm} (VII. 6.3.12)

Then we proceed by using the following algorithm:

a) \[ x_{n-k+1} = \frac{y^{(l-k+1)}}{n-k+1,n-k+1} \]

b) \[ x_k = (y_{l-k+1} - x_{n-k+1,n-k+1})/z_{l-k+1,k,k} \]  \hspace{1cm} (VII. 6.3.13)

c) \[ y^{(l-k+2)} = y^{(l-k+1)} - x_{k} z_k^* x_{n-k+1,n-k+1} \]

for all \( k = \lfloor \frac{n-1}{2} \rfloor \) where step (VII.6.3.13c) is not executed for \( k=1 \).

It can be seen that the computational cost for the solution of this linear system is a total number of \( 7 \lfloor \frac{n-1}{2} \rfloor + 2(1 + (-1)^n) \) time steps by using a maximum number of \( 2(n-1) \) processors.

Therefore, the complete factorisation of the coefficient matrix \( A \) and the solution of the linear system (VII.2.1.1) need either a total number of \( 21 \lfloor \frac{n-1}{2} \rfloor + \frac{9}{2}(1 + (-1)^n) \) time steps and a maximum number of \( 2(n-1)^2 \) processors or a total number of \( 22 \lfloor \frac{n-1}{2} \rfloor + \frac{9}{2}(1 + (-1)^n) \) time steps and a maximum number of \( (n-1)^2 \) processors.
CHAPTER VIII

NEW QUADRANT INTERLOCKING FACTORISATION METHODS
In this section we attempt to factorise a coefficient matrix \( A \) into the product of three matrices \( W, D \) and \( Z \) where:

\[
A = W . D . Z
\]  

(VIII.1.1.1)

The matrices \( W \) and \( Z \) are of the following form: if \( W_i \) and \( Z_i \) are the column vectors of the matrices \( W \) and \( Z^T \), then we have

\[
W = [W_1, W_2, \ldots, W_n]
\]  

and

\[
Z^T = [Z_1, Z_2, \ldots, Z_n]
\]  

(VIII.1.1.2)

where the column vectors \( W_i \)'s are of the following general form:

a) For \( n \) odd:

\[
W_i = \begin{cases} 
\begin{bmatrix} 0 & 0 & \ldots & 0 & 1_{w_{i+1},i} & \ldots & 1_{w_{n-i},i} & 0, \ldots, 0 \end{bmatrix}^T, & i = \left\lfloor \frac{n+1}{2} \right\rfloor \\
\begin{bmatrix} 0 & 0 & \ldots & 1 \end{bmatrix}^T, & i = \frac{n+1}{2} \\
\begin{bmatrix} 0 & 0 & \ldots & 0 & 1_{w_{i-1},i} & \ldots & 1_{w_{n-i+2},i} & 0, \ldots, 0 \end{bmatrix}^T, & i = \frac{n+3}{2} 
\end{cases}
\]  

(VIII.1.1.3)

b) For \( n \) even:

\[
W_i = \begin{cases} 
\begin{bmatrix} 0 & 0 & \ldots & 0 & 1_{w_{i+1},i} & \ldots & 1_{w_{n-i},i} & 0, \ldots, 0 \end{bmatrix}^T, & i = \left\lfloor \frac{n}{2} \right\rfloor - 1 \\
\begin{bmatrix} 0 & 1 & \ldots & 0 \end{bmatrix}^T, & i = \frac{n}{2} + 1 \\
\begin{bmatrix} 0 & 0 & \ldots & 0 & 1_{w_{i-1},i} & \ldots & 1_{w_{n-i+2},i} & 0, \ldots, 0 \end{bmatrix}^T, & i = \frac{n+2}{2} 
\end{cases}
\]  

(VIII.1.1.4)

The column vectors \( Z_i \)'s are of the following form:

\[
Z_i = \begin{cases} 
\begin{bmatrix} 0, \ldots, 0 & 1_{z_{i+1},i} & \ldots & 1_{z_{n-i+1},i} & 0, \ldots, 0 \end{bmatrix}^T, & i = \left\lfloor \frac{n+1}{2} \right\rfloor \\
\begin{bmatrix} 0, \ldots, 0 \end{bmatrix}^T, & i = \left\lfloor \frac{n+3}{2} \right\rfloor - 1 \\
\begin{bmatrix} 0, \ldots, 0 & 1_{z_{i+1},i} & \ldots & 1_{z_{n-i+1},i} & 0, \ldots, 0 \end{bmatrix}^T, & i = \frac{n+3}{2} 
\end{cases}
\]  

(VIII.1.1.5)

For \( n = 5 \), the matrices \( W \) and \( Z \) can be shown to have the following forms:
Similarly for n=6, we have

$$W = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

and

$$Z = \begin{bmatrix}
1 & z_{1,2} & z_{1,3} & z_{1,4} & z_{1,5} & z_{1,6} \\
0 & 1 & z_{2,3} & z_{2,4} & z_{2,5} & 0 \\
0 & 0 & 1 & z_{3,4} & 0 & 0 \\
0 & 0 & z_{4,3} & 1 & 0 & 0 \\
0 & z_{5,2} & z_{5,3} & z_{5,4} & 1 & 0 \\
z_{6,1} & z_{6,2} & z_{6,3} & z_{6,4} & z_{6,5} & 1
\end{bmatrix}$$

Note that the matrices \(W\) and \(Z\), as shown above, have the same structure as the \(W\) and \(Z\) matrices in the original Q.I.F. method, see Chapter VII except that the diagonal elements of \(Z\) are all unity. The matrix \(D\) is a diagonal matrix with its elements denoted by \(d_{i,i}\).

The computation of the elements of the matrices \(W, D\) and \(Z\) has a similar procedure to that of the Q.I.F. method, see section (VII.2.2) The process is carried out in \(\left\lceil \frac{n-1}{2} \right\rceil\) stages where, in general, at the \(i^{th}\) stage we have the solution of the \((2 \times 2)\) linear systems:
\[
\begin{align*}
\begin{cases}
  a_i, w_j, i \cdot a_{n-i+1}, j \cdot a_{j-i+1} = a_{j-i} \\
  a_i, a_{n-i+1}, j \cdot a_{n-i+1}, j = a_{j+n-i+1}
\end{cases}
\end{align*}
\]

\( j = i+1(1)n-i \) \hspace{1cm} (VIII.1.1.5)

to give the unknown quantities \( w_j, i, w_j, j = i+1(1)n-i \). Then,

we have the relationships:

\[
\begin{align*}
\begin{cases}
  z_{i, j} = a_{i, j}/a_{i, i} \\
  z_{n-i+1, j} = a_{n-i+1, j}/a_{n-i+1, n-i+1}
\end{cases}, j = i+1(1)n-i+1 \hspace{1cm} (VIII.1.1.6)
\end{align*}
\]

and

\[
\begin{align*}
\begin{cases}
  d_{i, i} = a_{i, i} \\
  d_{n-i+1, n-i+1} = a_{n-i+1, n-i+1}
\end{cases} \hspace{1cm} (VIII.1.1.7)
\end{align*}
\]

Finally, in preparation for the next stage we calculate the quantities:

\[
\begin{align*}
  a_{k, t} = a_{k, t}, a_{w, k, i}, d_{i, t} = a_{w, k, n-i+1}, d_{n-i+1, t}
\end{align*}
\]

\( k, t = i+1(1)n-i \).

Under the same assumption as before (in Chapter I) and the choice of the solution method (i.e., Cramer's rule, or Gauss-elimination method) for the \((2*2)\) linear systems in (VIII.1.5) according to the algorithms described in (VIII.3), we can obtain the following table (VIII.1.1) which illustrates the alternative computational costs of the factorisation process.

<table>
<thead>
<tr>
<th>Solution Method of the ((2*2)) Linear Systems</th>
<th>Total Time Steps</th>
<th>Maximum Number of Processors Working in Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cramer's Rule</td>
<td>(7[n-1] \quad 8[n-1] )</td>
<td>(\max(6(n-2), 2(n-2)^2)) (\max(6(n-2), (n-2)^2))</td>
</tr>
<tr>
<td>Gauss-elimination Method</td>
<td>(12[n-1] \quad 13[n-1] )</td>
<td>(2(n-2)^2) (\max(2(n-2), (n-2)^2))</td>
</tr>
</tbody>
</table>

\textbf{TABLE (VIII.1.1)}
SOLUTION OF LINEAR SYSTEMS

From (VIII.1.1) it can be seen that for the solution of the linear system $Ax=b$, it is sufficient and necessary to solve the following three simpler systems:

1. $Wy = b$  \hspace{1cm} (VIII.1.2.1)
2. $Du = y$  \hspace{1cm} (VIII.1.2.2)
3. $Zx = u$  \hspace{1cm} (VIII.1.2.3)

where $y$ and $u$ are two intermediate vectors for which the systems (VIII1.2.1) and (VIII1.2.2) are solved respectively.

The solution process of the system (VIII1.2.1) can be obtained from the following algorithm:

at stage $i$ for $i=1,2,...,\left\lfloor \frac{n-1}{2} \right\rfloor$, we compute

a)  \begin{align*}
    y_i &= b_i \\
    y_{n-i+1} &= b_{n-i+1}
\end{align*} \hspace{1cm} (VIII.1.2.4a)

and

b)  \begin{align*}
    b_j &= b_{j-w_j}y_i-w_jy_{n-i+1}, \quad j=i+1(1)n-i \hspace{1cm} (VIII.1.2.4b)
\end{align*}

We then proceed to the next stage.

Under the same assumption as before, in Chapter I, it can be seen that the solution of this system requires a total number of $3\left\lfloor \frac{n-1}{2} \right\rfloor$ time steps and a maximum number of $2(n-2)$ processors.

To solve the system (VIII.1.2.2) we have the following formula from which the quantities $u_i$, $i=1,2,...,n$ are evaluated:

\[ u_i = \frac{y_i}{d_{i,i}}, \quad i=1,2,...,n \] \hspace{1cm} (VIII.1.2.5)

The complete solution of the system requires 1 time step (one division) and a total number of $n$ processors.

To solve the system (VIII.1.2.3), we proceed as follows: if $n$ is odd, we have

\[ x_\ell = \hat{u}_\ell, \quad \ell=\frac{n+1}{2} \] \hspace{1cm} (VIII.1.2.6)

and

\[ \dot{u}_j = \hat{u}_j - x_\ell z_{i,\ell}, \quad j=1(1)n \text{ and } j\neq\frac{n+1}{2} \] \hspace{1cm} (VIII.1.2.7)
The rest of the elements of the vector $\mathbf{x}$ are found in pairs, by solving $\left(\frac{n-1}{2}\right)$ systems of $(2\times2)$ linear systems in $\left(\frac{n-1}{2}\right)$ distinct stages. In general, at the $i^{th}$ stage we solve the $2\times2$ linear system:

$$
\begin{align*}
\frac{x_i + z_i,n-i+1 x_{n-i+1} = u_i}{x_{n-i+1} + i x_i + x_{n-i+1} = u_i} & , i=1,2,\cdots,n. \quad (VIII.1,2.8)
\end{align*}
$$

for the unknowns $x_i$ and $x_{n-i+1}$. We then set:

$$
\begin{align*}
\frac{u_j = u_j,x_i x_j,i=n-i+1 + x_{n-i+1}}{j=1,2,\cdots,n-i+1} \quad (VIII.1,2.9)
\end{align*}
$$

If $n$ is even, then all the components of the vector $\mathbf{x}$ are found in pairs. To find all the pairs, the linear systems in (VIII.1.2.8) and the formula (VIII.1,2.9) are executed respectively for $i=1,2,\cdots,n$. It can be seen that the complete solution of this system requires a total number of $6\left[\frac{n-1}{2}\right]+2(2+(-1)^n)$ time steps and a maximum number of $3(n-2)$ processors if Cramer's rule is used for the solution of the $(2\times2)$ linear systems in (VIII.1.2.8) and a total number of $11\left[\frac{n-1}{2}\right]+7+4(-1)^n$ time steps with a maximum number of $2(n-2)$ processors if the Gauss-elimination method is used.

The following table illustrates the alternative computational costs for the complete factorisation of a coefficient matrix $A$ in the form $A=W.D.Z$ together with the computational cost for the solution of the linear system $A\mathbf{x}=\mathbf{b}$.

<table>
<thead>
<tr>
<th>Solution Method of the $(2\times2)$ Linear Systems</th>
<th>Total Time Steps</th>
<th>Maximum Number of Processors Working in Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cramer's Rule</td>
<td>$17\left[\frac{n-1}{2}\right]+2(2+(-1)^n)$</td>
<td>$\max{6(n-2),2(n-2)^2}$</td>
</tr>
<tr>
<td>Gauss-elimination elimination Method</td>
<td>$27\left[\frac{n-1}{2}\right]+7+4(-1)^n$</td>
<td>$2(n-2)^2$</td>
</tr>
<tr>
<td></td>
<td>$28\left[\frac{n-1}{2}\right]+7+4(-1)^n$</td>
<td>$\max{2(n-2),(n-2)^2}$</td>
</tr>
</tbody>
</table>

**TABLE (VIII.1,2.1)**
We now attempt to reduce a coefficient matrix $A$ to a matrix $X$ of the form,

$$X = \begin{bmatrix}
    x_{1,1} & 0 & \cdots & 0 & x_{1,n} \\
    x_{2,2} & \ddots & & & \vdots \\
    0 & \ddots & \ddots & & \vdots \\
    x_{n-1,2} & 0 & \ddots & \ddots & 0 \\
    x_{n,1} & \cdots & \cdots & \cdots & x_{n,n}
\end{bmatrix}.$$  \hspace{1cm} (VIII.2.1.1)

From the structure of the matrix $X$ we can see that the elements on the two main diagonals of the matrix $Q$ (say) containing the multipliers are zero, i.e.,

$$q_{i,i} = q_{n-i+1,n-i+1} = 0, \quad i=1,2,\ldots,n. \hspace{1cm} (VIII.2.1.2)$$

The elements $q_{i,j}, \quad i,j=1,2,\ldots,n$ are evaluated in a similar process to that of determining the elements $w_{i,j}$ in (VII.2.2) whilst the order in which the corresponding elements of the matrix $A$ are eliminated is similar to that of the Gauss-Jordan reduction method.

For the evaluation of the elements of the matrices $Q$ and $X$ we present the following algorithm:

At the start of the $i$\textsuperscript{th} stage, $i=1(1)\left\lfloor \frac{n-1}{2} \right\rfloor$ we have:

\begin{align*}
    x_{i,i} &= a_{i,i} \\
    x_{i,n-i+1} &= a_{i,n-i+1} \\
    x_{n-i+1,i} &= a_{n-i+1,i} \\
    x_{n-i+1,n-i+1} &= a_{n-i+1,n-i+1}
\end{align*} \hspace{1cm} (VIII.2.1.3)

and the solution of the $(2\times2)$ linear systems:

\begin{align*}
    x_{i,i}q_{j,i} + x_{n-i+1,i}q_{j,n-i+1} &= a_{j,i} \hspace{1cm} (VIII.2.1.4) \\
    x_{i,n-i+1}q_{j,i} + x_{n-i+1,n-i+1}q_{j,n-i+1} &= a_{j,n-i+1}
\end{align*}
for \( j = i + 1 \) to \( i \), to give the unknown quantities \( q_{j,i} \) and \( q_{j,n-i+1} \) for \( j = i + 1 \). Next, the modified \( a_{i,j} \)'s are determined from the following formulae:

\[
a_{k,l} = a_{k,l} - q_{k,i}a_{i,l} - q_{k,n-i+1}a_{n-i+1,l}, \quad k, l = (i+1)(n-1) - 1, 1.
\]

(VIII.2.1.5)

Then, we have the solution of the \((2 \times 2)\) linear systems:

\[
\begin{align*}
\begin{cases}
q_{j,i+1} + a_{i+1,j}q_{j,i+1} - a_{i+1,n-j}q_{j,n-j} &= a_{j,i+1} \\
a_{i+1,j}q_{j,i+1} + a_{j,i+1}q_{j,n-j} &= a_{j,n-j}
\end{cases}
\end{align*}
\]

(VIII.2.1.6)

for \( j = 1 \). to give the unknown quantities \( q_{j,i+1} \) and \( q_{j,n-j} \) for \( j = 1 \) and \( j = n(-1)^{n-i+1} \).

Finally, in preparation of the next stage we calculate the quantities:

\[
a_{k,l} = a_{k,l} - q_{k,i+1}a_{i+1,l} - q_{k,n-i}a_{n-i,l},
\]

for \( k = 1 \) to \( n(-1)^{n-i+1} \) and \( l = i + 1 \).

The above algorithm characterises the \( i \)th stage, \( i = 1, 2, \ldots, \left[ \frac{n-1}{2} \right] \), of the reduction process, however, to complete the evaluation of the elements of the matrix \( X \) we need to evaluate the element \( x_{\frac{n+1}{2}, \frac{n+1}{2}} \) in case of \( n \) odd from the first formula of (VIII.2.1.3) and \( x_{\frac{n}{2}, \frac{n}{2}}, x_{\frac{n}{2}, \frac{n}{2} + 1} \) in case of \( n \) even from (VIII.2.1.3). Note that the formula (VIII.2.1.7) is not executed at the \( \left[ \frac{n-1}{2} \right] \)th stage of the reduction process.

N.B. For \( n \) odd, the elements \( q_{j,i+1} \) where \( i = \left[ \frac{n-1}{2} \right] \) and \( j = 1 \) to \( n, j = \frac{n+1}{2} \) are evaluated from the formula:

\[
q_{j,i+1} = a_{j,i+1}/a_{i+1,i+1}.
\]

(VIII.2.1.8)

rather than from (VIII.2.1.6).

Under the same assumption as before, in Chapter I, and the
discussion on the two different ways of solving the (2*2) linear systems which are met during the reduction process (see section (VIII.2.3)) we represent the following table which illustrates the alternative computational costs of the reduction process of the Gauss-Jordan method for the Q.I.F. method.

<table>
<thead>
<tr>
<th>Solution Method of the (2*2) Linear Systems</th>
<th>Total Time Steps</th>
<th>Maximum Number of Processors Working in Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cramer's Rule</td>
<td>$12\left\lceil \frac{n-1}{2} \right\rceil - 4 + (-1)^n$</td>
<td>$\max{6(n-2), 2(n-2)^2}$</td>
</tr>
<tr>
<td></td>
<td>$14\left\lceil \frac{n-1}{2} \right\rceil - 5 + (-1)^n$</td>
<td>$\max{6(n-2), (n-2)^2}$</td>
</tr>
<tr>
<td>Gauss-elimination Method</td>
<td>$22\left\lceil \frac{n-1}{2} \right\rceil - 3 - \frac{7}{3}(1-(-1)^n)$</td>
<td>$2(n-2)^2$</td>
</tr>
<tr>
<td></td>
<td>$24\left\lceil \frac{n-1}{2} \right\rceil - 4 - \frac{7}{2}(1-(-1)^n)$</td>
<td>$\max{2(n-2), (n-2)^2}$</td>
</tr>
</tbody>
</table>

**TABLE (VIII.2.1.1)**

**VIII.2.2 THE EVALUATION OF THE R.H.S. VECTOR AND THE SOLUTION OF THE LINEAR SYSTEM**

In order to solve the linear system $\mathbf{A}\mathbf{y} = \mathbf{b}$ where the coefficient matrix $\mathbf{A}$ is reduced to a matrix $\mathbf{X}$ of the form shown in (VIII.2.1.1), the right hand side vector $\mathbf{b}$ has to be modified accordingly. This modification can be done by using the following algorithm:

a) $b_i = b_{i-k} - q_i, k \leq i < n-k, i = k+1(1)n-k$

and

b) $b_i = b_{i-k} - q_i, k+1 \leq i < n-k, i = 1(1)k \& i = n-k+1(1)n$

for $k=1(1)\left\lceil \frac{n-1}{2} \right\rceil$. However, in case of $n$ odd the formula (VIII.2.1a) is changed to the following relation:

$b_i = b_{i-k} - q_i, k = \left\lceil \frac{n-1}{2} \right\rceil$ and $i = 1(1)n$, $i \neq k$, (VIII.2.2)
It can be easily seen that the modification of the vector $\mathbf{b}$ requires a total number of $6\left[\frac{n-1}{2}\right] - (1+(-1)^n)$ time steps and a maximum number of $2(n-2)$ processors.

After the modification of the right hand side vector $\mathbf{b}$, the solution of the linear system:

$$x_\mathbf{y} = \mathbf{b}$$

(VIII,2,2,3)

can be regarded as the solution of $\left[\frac{n+1}{2}\right]$ sets of $(2\times2)$ linear equations of the form:

$$
\begin{cases}
  x_i, i \neq i, n-i+1, n-i+1 = b_i, \\
  x_{n-i+1, i} + x_{n-i+1, n-i+1} = b_{n-i+1}, \\
  x_{n-i+1, i} + x_{n-i+1, n-i+1} = b_{n-i+1}
\end{cases}
$$

(VIII,2,2,4)

Note that if $n$ is odd, then the central element of the vector $x_\mathbf{y}$, i.e. $y_{\frac{n+1}{2}}$ is evaluated from the formula:

$$\frac{y_{\frac{n+1}{2}}}{2} = \frac{b_{\frac{n+1}{2}}}{2} \cdot \frac{x_{\frac{n+1}{2}}}{2}$$

(VIII,2,2,5)

rather than from (VIII,2,2,4).

Since the solution of each of the above $(2\times2)$ linear systems is independent from the others, it can be seen that the complete solution of all the systems in (VIII,2,2,4) and hence the solution of linear system $x_\mathbf{y} = \mathbf{b}$ requires a total number of 3 time steps and a maximum number of $6\left[\frac{n+1}{2}\right]$ processors if Cramer's rule is used and a total number of 8 time steps and a maximum number of $2\left[\frac{n+1}{2}\right]$ processors if the Gauss-elimination method is used.

The following table illustrates the alternative computational costs for the complete Gauss-Jordan form of the quadrant interlocking factorisation method together with the computational cost of the solution of the linear system.
<table>
<thead>
<tr>
<th>Solution Method of the (2*2) Linear Systems</th>
<th>Total Time Steps</th>
<th>Maximum Number of Processors Working in Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cramer's Rule</td>
<td>$18\left[\frac{n-1}{2}\right]-2$</td>
<td>$\max{6(n-2),2(n-2)^2}$</td>
</tr>
<tr>
<td></td>
<td>$20\left[\frac{n-1}{2}\right]-3$</td>
<td>$\max{6(n-2),(n-2)^2}$</td>
</tr>
<tr>
<td>Gauss-elimination Method</td>
<td>$28\left[\frac{n-1}{2}\right]+\frac{1}{2}(1+5(-1)^n)$</td>
<td>$2(n-2)^2$</td>
</tr>
<tr>
<td></td>
<td>$30\left[\frac{n-1}{2}\right]-\frac{1}{2}(1-5(-1)^n)$</td>
<td>$\max{2(n-2),(n-2)^2}$</td>
</tr>
</tbody>
</table>

*TABLE (VIII.2.2.1)*
VIII.31 The Choleski Form of the Q.I.F. Method

We now present the Q.I.F. method in which a symmetric positive definite coefficient matrix $A$ is factorised such that:

$$A = WW^T$$  \hspace{1cm} (VIII.3.1.1)

where $W$ is of the form as follows: if $W_i$, $i=1,2,...,n$ is a column vector of $W$, then we have:

$$W = [W_1 \ W_2 \ ... \ W_n]$$  \hspace{1cm} (VIII.3.1.2)

where $W_i$, $i=1,2,...,n$ is of the following general form:

a) For $n$ odd,

$$W_i = \begin{cases} [0,0,...,0 \ w_{i,i},...,w_{n-i+1,i},0,...,0]^T, & i=1(1)\frac{n-1}{2} \\ [0,0,...,0 \ w_{i,i},0,...,0]^T, & i=\frac{n+1}{2} \\ [0,0,...,0 \ w_{n-i+2,i},...,w_{1,i},0,...,0]^T, & i=\frac{n+3}{2}(1)n. \end{cases}$$  \hspace{1cm} (VIII.3.1.3)

b) For $n$ even,

$$W_i = \begin{cases} [0,0,...,0 \ w_{i,i},...,w_{n-i+1,i},0,...,0]^T, & i=1(1)\frac{n}{2} \\ [0,0,...,0 \ w_{i,i},0,...,0]^T, & i=\frac{n+1}{2} \\ [0,0,...,0 \ w_{n-i+2,i},...,w_{1,i},0,...,0]^T, & i=\frac{n+2}{2}(1)n. \end{cases}$$  \hspace{1cm} (VIII.3.1.4)

Thus, for $n=5$ we have:

$$W = \begin{bmatrix} w_{1,1} & 0 & 0 & 0 & 0 \\ w_{2,1} & w_{2,2} & 0 & 0 & w_{2,5} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} \\ w_{4,1} & w_{4,2} & 0 & w_{4,4} & w_{4,5} \\ w_{5,1} & 0 & 0 & 0 & w_{5,5} \end{bmatrix}$$  \hspace{1cm} (VIII.3.1.5)

and for $n=6$, we have:

$$W = \begin{bmatrix} w_{1,1} & 0 & 0 & 0 & 0 & 0 \\ w_{2,1} & w_{2,2} & 0 & 0 & 0 & w_{2,6} \\ w_{3,1} & w_{3,2} & w_{3,3} & 0 & w_{3,5} & w_{3,6} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} & w_{4,6} \\ w_{5,1} & w_{5,2} & 0 & 0 & w_{5,5} & w_{5,6} \\ w_{6,1} & 0 & 0 & 0 & 0 & w_{6,6} \end{bmatrix}$$  \hspace{1cm} (VIII.3.1.6)
The elements of the matrix $W$ are evaluated in $k$ distinct stages where $k=[\frac{n+1}{2}]$. To compute the elements of the matrix $W$, we define at the start of the $k^{th}$ step, the matrix $A_k$, $k=1(1)[\frac{n-1}{2}]$, with elements $a_{i,j}^{(k)}$, $i,j=1(1)n$ as follows:

$$
A_1 = A
$$

$$
A_k = A - \sum_{i=1}^{k-1} W_i^*W_1^T - \sum_{i=n-k+2}^{n} W_i^*W_1^T, \quad k=2(1)[\frac{n-1}{2}].
$$

(VIII.3.2.1)

It can be observed that the first and last $(k-1)$ rows and columns of any $A_k$ matrix as defined by (VIII.3.2.1) are zero. Thus, by starting with $A_1=A$ we evaluate at the $k^{th}$ step, $k=1(1)[\frac{n-1}{2}]$, the elements which are non-zero in the $k^{th}$ and $(n-k+1)^{st}$ columns of $W$ by the following algorithm:

a) $w_{k,k} = \sqrt{a_{k,k}^{(k)}}$,

b) $w_{i,k} = a_{i,k}^{(k)}/w_{k,k}$, $i=k+1(1)n-k+1$

c) $w_{n-k+1,n-k+1} = \sqrt{a_{n-k+1,n-k+1}^{(k)}}/w_{n-k+1,n-k+1,k}$

(VIII.3.2.2)

d) $w_{i,n-k+1} = (a_{i,n-k+1}^{(k)}w_{i,n-k+1,k}^{(k)})/w_{n-k+1,n-k+1,k}$, $i=k+1(1)n-k$

e) $A_{k+1} = A_k - W_k^*W_k^T = W_{k+1,n-k+1}^*W_{n-k+1,k}^T$

To complete the evaluation of the $w_{i,j}$'s we need one more step in which, for $n$ odd, the $(\frac{n+1}{2})^{th}$ step is carried out to evaluate the centre element $w_{i,i}$ for $i=\frac{n+1}{2}$ from the formula (VIII.3.2.2a) and for $n$ even, the $\frac{n}{2}^{th}$ step is carried out to evaluate the elements $w_{k,k}$, $w_{k+1,k}$ and $w_{k+1,k+1}$ from the formulae (VIII.3.2.2a), (VIII.3.2.2b) and (VIII.3.2.2c).

Under the same assumption as before, in Chapter I, and that the time needed to perform a square root operation from either the Newton-
Raphson method or Heron's method is taken to be the equivalent of about 15 time steps, the following can be observed:-

1) Step (VII.3.2a) requires 15 time steps and 1 processor,
2) Step (VII.3.2b) needs 1 time step (one division) and a number of \((n-2k+1)\) processors,
3) Step (VII.3.2c) requires 17 time steps (one multiplication, one subtraction and one square root operation) and one processor,
4) Step (VII.3.2d) needs three time steps (one multiplication, one subtraction and one division) and \((n-2k)\) processors,
5) Step (VII.3.2e) requires a total number of three time steps, one multiplication with \(2(n-2k)^2\) processors and two subtractions with \(2(n-2k)^2\) processors. This step can alternatively be executed in four time steps, two multiplications with \((n-2k)^2\) processors and two subtractions with \((n-2k)^2\) processors.

Therefore, for the complete evaluation of the elements of the matrix \(W\) we need either a total number of \(15n+9\left\{\frac{n-1}{2}\right\}+\frac{3}{2}(1+(-1)^n)\) time steps with a maximum number of \(2(n-1)^2\) processors, or \(15n+10\left\{\frac{n-1}{2}\right\}+\frac{3}{2}(1+(-1)^n)\) time steps and a maximum number of \((n-1)^2\) processors.

**Example 1**

Consider the following (5*5) matrix as given by REINSCH & WILKINSON [50]

\[
\begin{bmatrix}
10 & 1 & 2 & 3 & 4 \\
1 & 9 & -1 & 2 & 3 \\
2 & -1 & 7 & 3 & -5 \\
3 & 2 & 3 & 12 & 1 \\
4 & 3 & -5 & -1 & 15
\end{bmatrix}
\]

then, the corresponding matrix \(W\) for the above example is:
Example 2

Consider the following (4*4) matrix as given by RUTISHAUSER [86],

\[
\begin{bmatrix}
5 & 4 & 1 & 1 \\
4 & 5 & 1 & 1 \\
1 & 1 & 4 & 2 \\
1 & 1 & 2 & 4 \\
\end{bmatrix}
\]

then, the corresponding matrix \( W \) for the above matrix is:

\[
\begin{bmatrix}
2.236068 & 0 & 0 & 0 \\
1.788854 & 1.337712 & 0 & 0.102598 \\
0.447214 & 0.078689 & 1.714986 & 0.923380 \\
0.447214 & 0 & 0 & 1.949359 \\
\end{bmatrix}
\]

VIII.3.3 SOLUTION OF THE LINEAR SYSTEM

For the solution of the linear system \( Ax = b \), where \( A = W \cdot W^T \), it is sufficient and necessary to solve the following two simpler systems:

\[
Wx = b \quad \text{(VIII.3.3.1)}
\]

and

\[
W^T \chi = \gamma \quad \text{(VIII.3.3.2)}
\]

The linear system (VIII.3.3.1) is first solved for the intermediate vector \( \chi \) with its components denoted by \( y_i, i = 1,2,\ldots,n \) and then the final solution \( x \) is obtained by solving the linear system (VIII.3.3.2).

The linear system (VIII.3.3.1) is solved for \( k = 1(1) \left[ \frac{n-1}{2} \right] \) by the following algorithm:

a) \( y_k = b^{(k)}/w_{k,k} \).
b) \( y_{n-k+1} = \frac{1}{w_{n-k+1}} \cdot (b_{n-k+1} - y_k \cdot w_{n-k+1}) \cdot w_{n-k+1}^{-1} \) \( (VIII.3.3.3) \)

c) \( b^{(k+1)} = b^{(k)} - y_k \cdot w_{k+1} - y_{n-k+1} \cdot w_{n-k+1} \)

If \( n \) is odd, step \((VIII.3.3.a)\) has to be carried out for \( k = \frac{n+1}{2} \) to find the element in the centre of the intermediate vector, \( y_k \), whilst if \( n \) is even, then steps \((VIII.3.3.a)\) and \((VIII.3.3.b)\) have to be executed for \( k = \frac{n}{2} \) to find the two centre elements \( y_{n/2} \) and \( y_{n/2+1} \) of the vector \( y \).

Under the same assumption as before, in Chapter I, it can be easily seen that for the complete evaluation of the components of the intermediate vector \( y \), either a total number of \( 7 \left[ \frac{n-1}{2} \right] + 1 + 5 \cdot 3 (-1)^n \) time steps and a maximum of \( 2(n-1) \) processors, or a total number of \( 8 \left[ \frac{n-1}{2} \right] + 1 + 5 \cdot 3 (-1)^n \) time steps and a maximum number of \( (n-1) \) processors are required.

To solve the linear system \((VIII.3.3.2)\) we shall use the algorithm presented by Evans & Hadjidimos \[49\], see also section \((VII.6.3)\), for the solution of the linear system \( Zx = y \), since the structure of the matrices \( Z \) and \( W^T \) is the same.

We first let

\[ W^T x = y = y^{(1)} \]

so that

\[ \sum_{i=1}^{n} x_i W_i = y^{(1)} \] \( (VIII.3.3.4) \)

where \( W_i \), \( i=1(1)n \) are the column vectors of the matrix \( W^T \), i.e.,

\[ W^T = \begin{bmatrix} W_1^* & W_2^* & \ldots & W_n^* \end{bmatrix} \] \( (VIII.3.3.5) \)

Hence, if we set \( i = \left[ \frac{n+1}{2} \right] \) then for \( n \) odd, the centre element of the vector \( x \), i.e. \( x_{n/2} \) can be evaluated as:

\[ x_{n/2} = y^{(1)} / \omega_{n/2} \] \( \omega_{n/2} \)

and

\[ y^{(2)} = y^{(1)} \cdot x_{n/2} W_{n/2}^* \] \( (VIII.3.3.6) \)

while for \( n \) even we find that
\[
\begin{align*}
\begin{cases}
x_{i+1} = y_{i+1}^{(1)}/w_{i+1, t+1}, \\
x_t = (y_t^{(1)} - x_{i+1} w_{t, l+1})/w_{t, t}
\end{cases}
\tag{VIII.3.3.7}
\end{align*}
\]

and
\[
y^{(2)} = y^{(1)} - x_k W^*_k - x_{k+1} W^*_{k+1},
\]

where \(w_{i,j}, i, j = 1, 2, \ldots, n\) denotes the elements of the matrix \(W^T\).

Then we proceed using the following algorithm:

\(a\) \(x_{n-k+1} = y_{n-k+1}^{(l-k+1)}/w_{n-k+1, n-k+1}\)

\(b\) \(x_k = (y_k^{(l-k+1)} - x_{n-k+1} W^*_{k, n-k+1})/w_{k, k}\)

\(c\) \(y^{(l-k+2)} = y^{(l-k+1)} - x_k W^* - x_{n-k+1} W^*_{n-k+1}\)

for all \(k = [n-1/2](-1)^l\) where step (VIII.3.3.8c) is not executed for \(k=1\).

It can be seen that the system (VIII.3.3.2) can be solved in a total number of \(7[n-1/2]+2(1+(-1)^n)\) time steps by using a maximum number of \(2(n-1)\) processors. Alternatively, we can solve this system in a total number of \(8[n-1/2]+2(1+(-1)^n)\) time steps by using a maximum number of \(2(n-1)\) processors.

Therefore, the complete factorisation of the coefficient matrix \(A\) and the solution of the linear system \(Ax=b\) requires either a total number of \(15n+23[n-1/2]+6+5(-1)^n\) time steps and a maximum number of \(2(n-1)^2\) processors, or a total number of \(15n+24[n-1/2]+6+5(-1)^n\) time steps and a maximum number of \((n-1)^2\) processors.
VIII.4 THE RELATIONSHIP BETWEEN THE CHOLESKI FORM OF THE Q.I.F. METHOD
AND THE CHOLESKI FACTORISATION METHOD

To show the relationship between these two methods, let us assume that by applying the Choleski factorisation method on a symmetric positive definite matrix $A$, we have obtained a matrix $L$ (say) such that:

$$A = L^*L^T$$  \hspace{1cm} (VIII.4.1)

where $L$ is a lower triangular matrix with its elements $l_{i,i}$, $i=1,2,...,n$ and $j=1,2,...,i$ defined by:

$$l_{i,i} = (a_{i,i} - \sum_{k=1}^{i-1} l_{k,i,k})^2$$  \hspace{1cm} (VIII.4.2)

and

$$l_{i,j} = (a_{i,i} - \sum_{k=1}^{j-1} l_{i,k,j,k})/(l_{j,j}), \quad j<i.$$

We also assume that the matrix $P$ with its elements denoted by $p_{i,j}$, $i,j=1,2,...,n$ is a permutation matrix with its non-zero elements defined as follows:

a) for $n$ odd

$$\begin{cases}
p(i,2i-1) = 1, & i = 1, (1) \frac{n+1}{2} \\
p(-\frac{n+1}{2}+i,n-2i+1) = 1, & i = 1, (1) \frac{n-1}{2}
\end{cases}$$  \hspace{1cm} (VIII.4.3a)

and

b) for $n$ even,

$$\begin{cases}
p(i,2i-1) = 1, & i = 1, (1) \frac{n}{2} \\
p(-\frac{n+1}{2}+i,n-2i) = 1, & i = 1, (1) \frac{n}{2}
\end{cases}$$  \hspace{1cm} (VIII.4.3b)

We can now prove the following theorem:

Theorem (VIII.4.1): In the Choleski form of the quadrant interlocking factorisation method of the matrix $PAP^T$ where $A$ is a symmetric positive definite matrix and $P$ a permutation matrix defined by (VIII.4.3a) and (VIII.4.3b) the resulting matrix $W$ which is of the form shown in (VIII.3.3) or (VIII.3.4) is a permuted form of the matrix $L$ where $L$ is the lower triangular matrix defined by (VIII.4.2) for which the relation (VIII.4.1) holds, i.e.,
\[ W = P L P^T \] \hspace{1cm} (VIII.4.4)

**Proof:**

It can be seen from (VI.4.1) that
\[ \text{PAP}^T = PL^T P T \] \hspace{1cm} (VIII.4.5)

It is evident by definition that
\[ P^T p = p P^T = I \] \hspace{1cm} (VIII.4.6)

hence,
\[ \text{PAP}^T = PLP^T P T \] \hspace{1cm} (VIII.4.7)

Since the matrix \( L \) is a lower triangular matrix and because of the definition of the matrix \( P \), it can be easily seen that \( PLP^T \)
is a matrix of the form given by (VIII.3.1.3) or (VIII.3.1.4). Therefore, by comparing (VIII.4.2) with (VIII.3.2.2) and the fact that:
\[ (PLP^T)^T = PL^T P T \] \hspace{1cm} (VIII.4.8)

we find that:
\[ W = PLP^T \] \hspace{1cm} (VIII.4.9)

and
\[ W^T = PL^T P T \] \hspace{1cm} (VIII.4.10)

hence we have
\[ \text{PAP}^T = W W^T \] \hspace{1cm} (VIII.4.11)

N.B. The same permutation matrix \( P \) can be used to show that the modified Q.I.F. method (EVANS & HADJIDIMOS [49]) when applied to \( \text{PAP}^T \) would give two matrices \( W \) and \( Z \) where
\[ \text{PAP}^T = W Z \] \hspace{1cm} (VIII.4.12)

and the \( W \) and \( Z \) matrices are permuted forms of two lower and upper triangular matrices \( L \) and \( U \), respectively, where \( L \) and \( U \) are obtained from the LU factorisation of the matrix \( A \), i.e.,
\[ W = PLP^T \] \hspace{1cm} (VIII.4.13)

and
\[ Z = P U P^T \] \hspace{1cm} (VIII.4.14)
VIII.5 THE ERROR ANALYSIS OF THE CHOLESKI FORM OF THE Q.I.F. METHOD

WILKINSON [43] has shown that for a symmetric positive matrix $A$ and a fixed point computation of the elements of the lower triangular matrix $L$ obtained from the Choleski factorisation of $A$, if

$$(n+1)2^{-t}||A^{-1}|| < 1 \quad (VIII.5.1)$$

and

$$|a_{i,j}| < (1-2^{-t}) \quad (VIII.5.2)$$

then all the computed elements $\hat{z}_{i,j}$ of the matrix $L$ satisfy $|\hat{z}_{i,j}| < 1$

and that

$$L^*L = A + E \quad (VIII.5.3)$$

where $E$ is a perturbation matrix with its elements defined as follows:

$$|\hat{z}_{i,j}| \leq \begin{cases} 
2^{-t-1}|\hat{z}_{i,i}|, & j > i \\
2^{-t-1}|\hat{z}_{i,i}|, & i > j \\
2^{-t}|\hat{z}_{i,i}|, & i = j 
\end{cases} \quad (VIII.5.4)$$

For a $(5*5)$ matrix, we have,

$$|E| \leq \frac{1}{2} 2^{-t} \begin{bmatrix} 2|\hat{z}_{1,1}| & |\hat{z}_{1,1}| & |\hat{z}_{1,1}| & |\hat{z}_{1,1}| & |\hat{z}_{1,1}| \\
|\hat{z}_{1,1}| & 2|\hat{z}_{2,2}| & |\hat{z}_{2,2}| & |\hat{z}_{2,2}| & |\hat{z}_{2,2}| \\
|\hat{z}_{1,1}| & |\hat{z}_{2,2}| & 2|\hat{z}_{3,3}| & |\hat{z}_{3,3}| & |\hat{z}_{3,3}| \\
|\hat{z}_{1,1}| & |\hat{z}_{2,2}| & |\hat{z}_{3,3}| & 2|\hat{z}_{4,4}| & |\hat{z}_{4,4}| \\
|\hat{z}_{1,1}| & |\hat{z}_{2,2}| & |z_{3,3}| & |z_{4,4}| & 2|\hat{z}_{5,5}| \end{bmatrix}$$

and the generating structure of the error matrix $P$ can be illustrated as:

![Diagram](FIGURE (VIII.5.1))
where the number (i) on a line denotes that the corresponding row or column of the matrix $E$ is dominated by the diagonal element $e_{i,i}$.

From the relationship between the Choleski form of the Q.I.F. method and the Choleski factorisation method as shown in section (VIII.4) and the structure of the permutation matrix $P$ given by (VIII.4.3) we can easily find a similar structure of the perturbation matrix $E$ for the Choleski form of the Q.I.F. method. That is, if the conditions given by (VIII.1) and (VIII.2) hold for a symmetric positive definite matrix $A$ then all the computed elements of the matrix $W$ in (VIII.3.1.3) or (VIII.3.1.4) satisfy $|w_{i,i}| < 1$ and that

$$W W^T = A + E$$  \hspace{1cm} (VIII.5.6)

where the elements $e_{i,j}$, $i,j=1,2,\ldots,n$ of the error matrix $E$ are defined as follows:

a) for $n$ odd we have:

$$
\begin{align*}
|e_{i,j}| &= |e_{j,i}| = 2^{-t-1}|w_{j,j}|, & i &= j+1(1)n-j+1, \\
|e_{i,n-j+1}| &= |e_{n-j+1,i}| = 2^{-t-1}|w_{n-j+1,n-j+1}|, & i &= j+1(1)n-j, \\
|e_{j,j}| &= 2^{-t}|w_{j,j}|, & j &= 1(1)n
\end{align*}
$$  \hspace{1cm} (VIII.5.7)

b) for $n$ even, we have:

$$
\begin{align*}
|e_{i,j}| &= |e_{j,i}| = 2^{-t-1}|w_{j,j}|, & i &= j+1(1)n-j+1, \\
|e_{i,n-j+1}| &= |e_{n-j+1,i}| = 2^{-t-1}|w_{n-j+1,n-j+1}|, & i &= j+1(1)n-j, \\
|e_{j,j}| &= 2^{-t}|w_{j,j}|, & j &= 1(1)n
\end{align*}
$$  \hspace{1cm} (VIII.5.8)

Similarly the generating structure of the error matrix $E$ for the Choleski form of the Q.I.F. method compared with Figure (VIII.5.1) is illustrated as:
a) for $n$ odd:

\[
\begin{array}{cccc}
1 & 2 & \cdots & \frac{(n-1)/2}{(n+1)/2} \\
\frac{(n-1)/2}{(n+1)/2} & \frac{(n+1)/2}{(n+3)/2} & \cdots & n-1 \\
\frac{(n+1)/2}{(n+3)/2} & \frac{(n+3)/2}{(n+5)/2} & \cdots & n \\
n-1 & n & \cdots & 1
\end{array}
\]

and

b) for $n$ even:

\[
\begin{array}{cccc}
1 & 2 & \cdots & \frac{n/2}{n/2+1} \\
\frac{n/2}{n/2+1} & \frac{n/2+1}{n/2+2} & \cdots & n-1 \\
\frac{n/2+1}{n/2+2} & \frac{n/2+2}{n/2+3} & \cdots & n \\
n-1 & n & \cdots & 1
\end{array}
\]

where the number (i) on a line denotes that the corresponding row or column of the matrix $E$ is dominated by $|w_{i,i}|$. 


[37] ZLATER, Z. and WASNIEWSKI, J., "Package Y12M: Solution of Large and Sparse Systems of Linear Algebraic Equations", Mathematics Institute, University of Copenhagen, Denmark.


[49] EVANS, D.J. and HADJIDIMOS, A., "A Modification to the Quadrant Interlocking Factorisation Parallel Method",


[80] RUTISHAUSER, H., "Solution of Eigenvalue Problem with the LR-Transformation" in "Further Contribution to the Solution of Simultaneous Linear Equations and Determination of Eigenvalues"

APPENDIX I

FORTRAN PROGRAMS ON ELIMINATION METHODS FOR THE SOLUTION OF EIGENSYSTEMS
subroutine muller(n, a, s, n, r, n, l, b, sbr, rbr, eigen, bb, 1 c, w, v, no, mflag, iflag, ha, pivot)

double precision a(nn), f(3), v(3), h1, h2, pesy1, pesy2, g, b(n), 1 delta, mach, den, pp, det, eigen(n), c(n), bb(n), xmax, w(n), s, s1, 2 s2, lambda, aflag(8), pivot(n)

integer z, snr(nn), rnr(nn1), sbr(z), rbr(z), ha(n, 11), iflag(10)

*********************************************************************
* * THIS SUBROUTIN EVALUATES A PRESCRIBED NUMBER OF EIGENVALUES *  *
* USES AND/OR THEIR CORRESPONDING EIGENVECTORS OF AN(N*N) MATRIX *  *
* AND 3 SPARSE GAUSSIAN ELIMINATION METHOD IS USED WITH *  *
* PIVOTAL STEAGRIETY IN ORDER TO EVALUATE THE DETERNINANT *  *
* OF THE MATRIX (A-LAMBDA*I) FOR VARIOUS VALUES OF LAMBDA *  *
* WHERE WE ATTEMPT TO MINIMISE THE NUMBER OF NON-ZERO ELEMENTS CREATED DURING THE ELIMINATION PROCESS AND TO OBTAIN *  *
* ACCURACY IN OUR COMPUTATIONS *  *
*
*********************************************************************

INITIALISATION OF THE PARAMETERS

N - THE SIZE OF THE MATRIX A FOR WHICH THE EIGENVALUES ARE EVALUATED.

Z - THE NUMBER OF NON-ZERO ELEMENTS IN THE MATRIX A INCLUDING THE ZEROS ON THE DIAGONAL OF THE MATRIX A.


A = L*U.

THIS ARRAY IS OF THE SIZE NN GIVEN BY THE USER AND IT HAS TO BE LARGE ENOUGH TO STORE ALL THE NON-ZERO ELEMENTS OF THE MATRIX A AFTER THE ELIMINATION. THE SUGGESTED VALUE FOR NN IS BETWEEN 2Z AND 3Z.


ARRAY RNR - THE INTEGER ARRAY RNR IS OF THE SIZE NN1 GIVEN BY

NOTE THAT, IF IN THE ORIGINAL MATRIX A, THERE EXISTS A ZERO DIAGONAL ELEMENT THEN THIS VALUE MUST BE STORED AND THEREFORE IS NOT CONSIDERED AS A ZERO ELEMENT.

ARRAY B - THE ONE-DIMENSIONAL ARRAY B IS OF THE SIZE Z AND OF THE TYPE DOUBLE PRECISION. THIS ARRAY IS USED AS A WORK AREA.

ARRAYS SBR AND RBR - THESE TWO INTEGER ARRAYS ARE OF THE SIZE Z AND ARE WORKING SPACE.


ARRAY BB - THIS ARRAY IS OF THE TYPE DOUBLE PRECISION AND HAS N COMPONENTS. THE EIGENVECTOR CORRESPONDING TO THE LATEST EIGENVALUE EVALUATED IS STORED IN THIS ARRAY. HOWEVER, ALL THE EIGENVECTORS REQUIRED WILL BE PRINTED BY THE SUBROUTINE AUTOMATICALLY.

ARRAYS CAND W - THESE TWO ARRAYS ARE OF THE TYPE DOUBLE PRECISION AND OF THE SIZE N WHICH ARE USED AS WORKING SPACE.

ARRAY V - THE DOUBLE PRECISION ARRAY V HAS 3 LOCATIONS AND CONTAINS THE THREE INITIAL GUESSES OF ANY EIGENVALUE. THESE VALUES ARE GIVEN BY THE USER BEFORE CALLING THE ROUTINE AND CAN BE THE SAME FOR ALL THE EIGENVALUES REQUIRED.

NO - THE INTEGER VARIABLE NO DENOTES THE NUMBER OF EIGENVALUES REQUIRED AND IS INITIALIZED BY THE USER.

MFLAG - THE USE OF THIS INTEGER VARIABLE IS DESCRIBED AS FOLLOWS.

MFLAG=1 DENOTES THAT ONLY THE EIGENVALUES ARE REQUIRED TO BE EVALUATED.

MFLAG=2 DENOTES THAT THE EIGENVALUES AND EIGENVECTORS ARE BOTH REQUIRED.

MFLAG=3 DENOTES THAT THE IMPROVED EIGENVALUES BY THE INVERSE ITERATION ARE REQUIRED.

MFLAG=4 DENOTES THAT THE IMPROVED EIGENVALUES AND EIGENVECTORS ARE REQUIRED.

ARRAY IFLAG - THE USE OF THIS INTEGER ARRAY OF THE SIZE IFLAG(10) IS DESCRIBED IN THE SUBROUTINE DETER.F.
ARRA HYA _THY INTEGER ARRAY HA IS OF THE SIZE (N*I). THE USE OF THIS ARRAY IS EXPLAINED IN THE ROUTIN DETER.

ARRAY PIVOT _THE DOUBLE PRECISION ARRAY PIVOT IS OF THE SIZE N AND CONTAINS THE PIVOTAL ELEMENTS OF THE MATRIX (A-LAMBDA*I) AFTER EACH ELIMINATION PROCESS. THESE VALUES CAN BE CHECKED BY THE USER.

***************************************************************************************
iz=z
iha=n
10 do i=1,n
   c(i)=1.0d0
10    do i=1,z
      b(i)=a(i)
      sbr(i)=snr(i)
      rbr(i)=rnr(i)
20  continue
30 write(6,30)
30 format('THE GIVEN DATA IS AS FOLLOWS :_')
35 format('RNR ',' ,5x,' SNR ',' ,5x,' A ')
50 do i=1,z
60    write(6,40) rbr(i),sbr(i),b(i)
40 format(i6,5x,i6,5x,e30.20)
60    continue
390 ii=1,nn1
   do 110 i=ii,nn1
   a(i)=0.0d0
   snr(i)=0
60  continue
110 z=iz
   do 70 j=1,z
      a(j)=b(j)
      rnr(j)=rbr(j)
      snr(j)=sbr(j)
70  continue
   do 80 j=1,z
      if(rnr(j).ne.snr(j)) goto 60
      a(j)=a(j)-v(i)
80  continue
   call deter(nn,z,a,snr,nn,rnr,nn1,pivot,ha,iha,aflag,iflag,ifail,det)
      /1 f(i)=det
   z=iz
   if(det.ne.0.0d0) goto 90
      v(3)=v(i)
90  goto 205
f(i)=det
   if(ii.lt.2) goto 110
   den=1.0d0
   do 100 ip=1,ii-1
      den=den*(v(i)-eigen(ip))
100 continue
f(i)=det/den
continue
110 h1=v(2)-v(1)
h2=v(3)-v(2)
pesy1=h2/h1
delta=1.0d0+pesy1
g=f(1)*pesy1-f(2)*delta*delta+f(3)*pesy1*delta+f(3)
pp=4.*f(3)*delta*pesy1*(f(1)*pesy1-f(2)*delta+f(3))
if(g*g-pp.lt.0.0d0) goto 140
if(g.lt.0.0d0) goto 130
mach=g+sqrt(g*g-pp)
goto 150
130 mach=g-sqrt(g*g-pp)
goto 150
140 mach=g
150 pesy2=-2.0d0*f(3)*delta/mach
v(1)=v(2)
v(2)=v(3)
v(3)=v(3)+pesy2*h2
if(dabs(v(3)-v(2)).lt.0.0000001) goto 205
f(1)=f(2)
f(2)=f(3)
do 160 i=1,nn1
a(i)=0.0d0
snr(i)=0
rnr(i)=0
160 continue
z=iz
do 170 i=1,z
a(i)=b(i)
rnr(i)=rbr(i)
snr(i)=sbr(i)
170 continue
do 180 i=1,z
if(rnr(i).ne.snr(i)) goto 120
a(i)=a(i)-v(3)
180 continue
call deter(n,z,a,snr,nn,rnr,nn1,pivot,ha,iha,aflag,iflag,ifail,det)
z=iz
f(3)=det
if(det.eq.0.0d0) goto 205
if(ii.lt.2) goto 200
den=1.0d0
do 190 ip=1,ii-1
den=den*(v(3)-eigen(ip))
190 continue
f(3)=det/den
goto 120
200 goto 120
205 write(6,210) v(3)
210 format('EIGEN VALUE=',.e25.15)
eigen(ii)=v(3)
ifail=1
if(mflag.eq.1) goto 390
do 220 jj=1,nn
bb(jj)=c(jj)
220 continue
221 call y12mdf(n,a,nn,bb,pivot,snr,ha,iha,iflag,ifail)
222 xmax=0.0d0
223 do 230 jj=1,n
224 if(dabs(bb(jj)) .lt. dabs(xmax)) goto 230
225 xmax=bb(jj)
226 continue
227 do 240 jj=1,n
228 bb(jj)=bb(jj)/xmax
229 continue
230 if(mflag.eq.3) goto 255
231 write(6,245)
232 format('THE EIGENVECTOR CORRESPONDING TO THE ABOVE EIGENVALUE')
233 write(6,250)(bb(jj),jj=1,n)
234 format(e30.20)
235 continue
236 do 260 is=1,nn1
237 a(is)=0.0d0
238 rnr(is)=0
239 snr(is)=0
240 continue
241 do 270 is=1,z
242 a(is)=b(is)
243 rnr(is)=rbr(is)
244 snr(is)=sbr(is)
245 continue
246 if(mflag.eq.2) goto 390
247 do 290 js=1,n
248 s=0.0d0
249 do 280 is=1,z
250 if(snr(is).eq.js) s=s+a(is)*bb(rnr(is))
251 continue
252 w(js)=s
253 continue
254 s1=0.0d0
255 do 300 is=1,n
256 s1=s1+w(is)*bb(is)
257 continue
258 s2=0.0d0
259 do 310 is=1,z
260 s2=s2+bb(is)*bb(is)
261 continue
262 lambda=s1/s2
263 write(6,320) lambda
264 format('THE IMPROVED EIGEN VALUE=',e30.20)
265 if(mflag.eq.3) goto 390
266 do 330 is=1,z
267 if(rnr(is).ne.snr(is)) goto 330
268 a(is)=a(is)-lambda
269 continue
270 call deter(n,z,a,snr,nn,rnr,nn1,pivot,ha,iha,iflag,iflag
271 1 ifail,del)
272 z=iz
273 do 340 is=1,n
274 bb(is)=c(is)
275 continue
call y12mdf(n,a,nn,bb,pivot,snr,ha,iha,iflag,ifail)
  xmax=0.0d0
  do 350 jj=1,n
  if(dabs(bb(jj)).lt.dabs(xmax)) goto 350
  xmax=bb(jj)
  350 continue
  do 360 jj=1,n
  bb(jj)=bb(jj)/xmax
  360 continue
  write(6,370)
  370 format('THE IMPROVED EIGEN VECTOR BY INVERSE ITERATION')
  write(6,380)
  380 format('******************************************************************************')
  write(6,250) (bb(jj),jj=1,n)
  390 continue
  return
end
**DETER.F PAGE 1**

SUBROUTINE DETER(N,Z,A,SNR,NN,RNR,NN1,PIVOT,HA,IHA,AFLAG,  
1 IFLAG,IFAIL,DET)  
IMPLICIT DOUBLE PRECISION (A-B,G,P,T-V),INTEGER (C,F,  
1 H-N,R-S,Z)  
DOUBLE PRECISION A(NN),PIVOT(N),AFLAG(S),DET  
INTEGER SNR(NN),RNR(NN1),HA(IHA,11),IFLAG(10)

******************************************************************************
* THIS SUBROUTINE EVALUATES THE DETERMINANT OF A GIVEN MATRIX (I.E., MATRIX DEFINED BY (A,SNR,RNR)). THIS EVALUATION IS ACHIEVED BY FIRSTLY REDUCING THE MATRIX INTO AN UPPER TRIANGULAR MATRIX AND THEN MULTIPLYING THE DIAGONAL ELEMENTS OF THE UPPER TRIANGULAR MATRIX. THE TOTAL NUMBER OF INTERCHANGES MADE DURING THE REDUCTION PROCESS IS DETERMINED AND STORED IN MARK1. *
******************************************************************************

**INITIALISATION OF THE PARAMETERS**

THE PARAMETERS N,Z,NN,NN1 AND PIVOT ARE DESCRIBED IN THE ROUTINE MULLER.

ARRAY A THIS ARRAY CONTAINS THE NON-ZERO ELEMENTS OF THE MATRIX A BEFORE AND AFTER THE ELIMINATION PROCESS. NOTE THAT THE VALUE (LAMDA) WHICH IS A GEUSS TO AN EIGENVALUE HAVE ALREADY BEEN SUBTRACTED FROM THE DIAGONAL ELEMENTS OF THE ORIGINAL MATRIX A GIVEN BY THE USER IN THE ROUTINE MULLER.

ARRAYS SNR AND RNR THE COLUMN AND ROW POSITIONS OF THE NON-ZERO ELEMENTS OF THE MATRIX (A-LAMDA*I) ARE STORED IN THE ARRAYS SNR AND RNR RESPECTIVELY IN THE SAME ORDER AS THEY ARE STORED IN THE ARRAY A.

ARRAY HA INTEGER ARRAY OF LENGTH HA(IHA,11). WORK SPACE. INFORMATION ABOUT ROW STARTS ROW ENDS, COLUMN STARTS AND COLUMN ENDS WILL BE STORED BY THE ROUTINE Y12MBF IN ARRAY HA. SOME OTHER INFORMATION ABOUT THE ROWS AND THE COLUMNS IS ALSO KEPT IN HA.

IHA AN INTEGER WHICH IS INITIALISED IN THE ROUTINE MULLER.

ARRAY AFLAG THIS ARRAY IS OF THE TYPE DOUBLE PRECISION AND OF THE LENGTH AFLAG(S).A.THE FIRST FOUR COMPONENTS OF THIS ARRAY CAN BE INITIALISED BY THE USER IN ORDER TO IMPROVE THE PERFORMANCE OF THE ROUTINE Y12MJF CONSIDERABLY.THE OTHER COMPONENTS CAN (OPTIONALLY) BE USED AS OUTPUT PARAMETERS IN
ORDER TO OBTAIN SOME INFORMATION ABOUT THE ELIMINATION PROCESS.

AFLAG(1) - THIS REPRESENTS THE STABILITY FACTOR THAT IS AN ELEMENT OF THE MATRIX A CAN BE CHOSEN AS A PIVOT ONLY IF THIS ELEMENT IS SMALLER THAN THE RATIO OF THE ABSOLUTE VALUE OF THE LARGEST ELEMENT IN ITS ROW AND THE STABILITY FACTOR. AFLAG(1) MUST BE SELECTED. SMALLER VALUES (BUT LARGER THAN ONE) OF AFLAG(1) MAY IMPROVE THE ACCURACY OF THE COMPUTATIONS BUT THIS CHOICE MAY CAUSE MANY FILL-INS. IN THIS SUBROUTINE WE HAVE SET AFLAG(1) = 16.

AFLAG(2) - DROP-TOLERANCE. I.E. IF AFLAG(2) > 0 THEN THE ROUTINE Y12MJF WILL REMOVE ALL THE ELEMENTS WHICH BECOME SMALLER IN ABSOLUTE VALUE THAN AFLAG(2) DURING THE COMPUTATIONS. STORAGE AND TIME MAY BE SAVED IN THIS WAY. THE USE OF POSITIVE VALUES OF AFLAG(2) CAN SOMETIMES PREVENT UNDERFLOWS. IF A POSITIVE VALUE OF AFLAG(2) IS USED AND IF THE MATRIX A IS SINGULAR THEN THE SINGULARITY IS VERY OFTEN DETECTED. THE PARAMETER MUST BE USED CAREFULLY BECAUSE THE ACCURACY MAY BE REDUCED. IN THIS SUBROUTINE AFLAG(2) = 1.E-12 HAS BEEN INITIALISED.

AFLAG(3) - THIS PARAMETER IS USED IN THE SUBROUTINE Y12MJF. Y12MJF STOPS THE COMPUTATION AND GIVES AN ERROR DIAGNOSTIC IF SOME ELEMENT OF THE MATRIX BECOMES larger IN ABSOLUTE VALUE THAN AFLAG(3). OVERFLOWS IN MATRIX A ARE PREVENTED BY THE USE OF THIS PARAMETER. IN THIS ROUTINE THE VALUE OF AFLAG(3) = 1.E+16 HAS BEEN INITIALISED.

AFLAG(4) - THE ROUTINE Y12MJF STOPS THE COMPUTATION IF THE ABSOLUTE VALUE OF THE CURRENT PIVOTAL ELEMENT IS LESS THAN AFLAG(4) * AFLAG(6). IN THIS WAY THE USE OF SMALL POSITIVE VALUES OF AFLAG(4) IS ANOTHER MEANS TO DETECT SINGULARITY; BUT THE VALUES OF AFLAG(4) MUST BE CHOSEN VERY CAREFULLY. IN THIS SUBROUTINE AFLAG(4) = 1.D-30 HAS BEEN INITIALISED.

AFLAG(5) - GROWTH FACTOR. AFTER EACH STEP OF THE ELIMINATION THE ROUTINE Y12MJF SETS AFLAG(5) = AFLAG(7) / AFLAG(6). A VERY LARGE VALUE OF AFLAG(5) INDICATES THE POSSIBILITY OF APPRECIABLE ERROR IN THE COMPUTED SOLUTION FOR EVALUATING THE EIGENVECTOR. INCREASING THE VALUE OF AFLAG(1) TENDS TO REDUCE THIS ERROR.

AFLAG(6) - THE LARGEST VALUE OF THE ORIGINAL MATRIX ON ENTRY IS STORED BY THE ROUTINE Y12MBF IN AFLAG(6). UNCHANGED ON EXIT.

AFLAG(7) - THE LARGEST ELEMENT FOUND AT ANY STAGE OF THE ELIMINATION WILL BE STORED BY Y12MJF IN AFLAG(7).

AFLAG(8) - THE MINIMUM PIVOTAL ELEMENT IN ABSOLUTE VALUE WILL BE
STORED IN AFLAG(8) BY THE ROUTINE Y12Mjf.

ARRAY IFLAG _INTEGER ARRAY OF LENGTH IFLAG(10). SOME OF THE COM- ONENTS OF THE ARRAY ARE USED TO IMPROVE THE PERFORMANCE OF THE SUBROUTINE. THE OTHERS CAN OPTIONALLY BE USED AS OUTPUT PARAMETERS.

IFLAG(1) ON ENTRY TO THE ROUTINE Y12Mjf AS ON EXIT OF THE ROU- TINE Y12MBF.

IFLAG(2) AT EACH STAGE OF THE ELIMINATION THE PIVOTAL ELEMENT IS SOUGHT IN THE IFLAG(2) ROWS WHICH CONTAIN THE LEAST NUMBER OF NON_ZERO ELEMENTS. THE CHOICE OF IFLAG(2)=1,2 OR 3 IS RECOMMENDED BY [37]. UNCHANGED ON EXIT.

IFLAG(3) IN THE GENERAL CASE IFLAG(3)=1 SHOULD BE INITIALISED. IF MATRIX A IS SYMMETRIC AND DEFINITE (EITHER POSITIVE OR NEGATIVE) OR DIAGONALLY DOMINANT THEN IFLAG(3)=2 IS THE BEST CHOICE; HOWEVER IF THE NON_ZERO ELEMENTS ARE NOT FAR FROM THE MAIN DIAGONAL THEN IFLAG(3)=0 WILL PERFORM VERY WELL. SEE[37].

IFLAG(4) THIS VARIABLE IS SET BY THE ROUTINE Y12MBF AND IS UNCHANGED ON EXIT.


IFLAG(6) THE NUMBER OF COLLECTIONS IN THE ROW ORDERED LIST (ARRAY SNR) WILL BE STORED BY THE ROUTINE Y12Mjf IN IFLAG(6). IF THIS NUMBER IS LARGE THEN IT IS BETTER TO CHOOSE A LARGER VALUE OF NN WITH THE NEXT CALL OF THE SUBROUTINE FOR THE SAME OR SIMILAR MATRIX.

IFLAG(7) THE NUMBER OF COLLECTIONS IN THE COLUMN ORDERED LIST (ARR- AY RNR) WILL BE STORED BY THE ROUTINE Y12Mjf IN IFLAG(7). IF THIS NUMBER IS LARGE THEN IT IS BETTER TO CHOOSE A LARGER VALUE OF NNI DURING THE NEXT CALL OF THE ROUTINE FOR THE SAME OR SIMILAR MATRIX.

IFLAG(8) THE MAXIMUM NUMBER OF THE NONZERO ELEMENTS KEPT IN ARRAY
A at any stage of the elimination process will be stored by the routine Y12MJF in IFLAG(8). If this number is much smaller than NN then the value of NN may be reduced during the next call of the routine with the same or smaller matrix.

IFLAG(9). If IFLAG(4)=1 then the minimum value of NN1 which may be used in the evaluation of the eigenvectors of the same matrix at the next call of the routine will be stored in IFLAG(9).

NB

All the parameters must be initialised within this routine.

ERROR DIAGNOSTICS

ERROR IFAIL=3 A pivotal element A(I,J,K) (AFLAG(4)*AFLAG(6) is found and the matrix is declared to be numerically singular.

ERROR IFAIL=4 AFLAG(5) is larger than AFLAG(3).

ERROR IFAIL=5 NN is small.

ERROR IFAIL=6 NN1 is small.

ERROR IFAIL=7 A row without a non-zero element is found.

ERROR IFAIL=8 A column without a non-zero element is found.

ERROR IFAIL=9 A pivotal element equal to zero is found.

DO 10 I=1,IHA
DO 10 J=1,11
HA(I,J)=0
10 CONTINUE
DO 20 I=1,8
AFLAG(I)=0.0DO
DO 30 I=1,10
30 IFLAG(I)=0
AFLAG(1)=16.0DO
AFLAG(2)=1.D-12
AFLAG(3)=1.D+16
AFLAG(4)=1.D-30
IFLAG(2)=1
IFLAG(3)=1
IFLAG(4)=1
IFLAG(5)=1
IFAIL=1
CALL Y12MBF(N,Z,A,SNR,NN,RNR,NN1,HA,IHA,AFLAG,IFLAG,IFAIL)
IFAIL=1
MARK1=0
CALL Y12MJF(N,Z,A,SNR,NN,RNR,NN1,PIVOT,HA,IHA,AFLAG,IFLAGE,1
IFAIL,MARK1)
DET=1.0DO
DO 40 I=1,N
DET=DET*PIVOT(I)
40 CONTINUE
MARK2=(-1)**MARK1
IF(MARK2.GE.0) GOTO 50
DET=-DET
50 RETURN
END
SUBROUTINE SECANT(N,Z,A,SNR,NN,RNR,NN1,B,SBR,RBR,EIGEN,BB,
C,W,V,NO,I,FLAG,H,PIVOT)
C
DOUBLE PRECISION A(NN),F(3),V(2),LAMBDA,B(Z),EIGEN(N),C(N),
1 W(N),BB(N),AFLAG(8),XMAX,DET,DEN,S,S1,S2
C
INTEGER Z,SNR,NN,RNR,NN1,B,SBR,RRZ,HA(N,11),IFLAG(10)
C
**************************************************************************
**
* THIS SUBROUTINE EVALUATES A PRESCRIBED NUMBER OF *
* EIGENVALUES AND/OR THEIR CORRESPONDING EIGENVECTORS OF AN *
* (N*N) MATRIX A BY THE SECANT METHOD AS DESCRIBED IN *
* CHAPTERS 2 AND 3. SPARSE GAUSSIAN ELIMINATION METHOD IS *
* USED WITH PIVOTAL STRATEGIES IN ORDER TO EVALUATE THE DE-
* TERMINANT OF THE MATRICES (A-LAMBDA*I) FOR VARIOUS VALUES *
* OF LAMBDA WHERE WE ATTEMPT TO MINIMISE THE NUMBER OF NON-
* ZERO ELEMENTS CREATED DURING THE ELIMINATION PROCESS AND *
* ALSO TO MAINTAIN ACCURATE COMPUTATION *
*
**************************************************************************

INITIALISATION OF THE PARAMETERS

INITIALISATION OF ALL THE PARAMETERS USED IN THIS ROUTINE
ARE DESCRIBED IN THE SUBROUTINES MULLER AND DETER, EXCEPT:

ARRAY V - THE DOUBLE PRECISION ARRAY V OF THE LENGTH V(2)
SHOULD CONTAIN THE TWO INITIAL GUESSES TO THE EI-
GENVALUES. THESE TWO GUESSES ARE MADE AND PASSED
THROUGH BY THE USER.

**************************************************************************
IHA=N
DO 10 I=1,Z
B(I)=A(I)
RBR(I)=RNR(I)
SBR(I)=SNR(I)
10 CONTINUE
DO 20 I=1,N
C(I)=1.0DO
20 CONTINUE
IZ=Z
WRITE(6,30)
30 FORMAT('THE GIVEN DATA IS AS FOLLOWS:
',E30.20)
DO 40 I=1,Z
WRITE(6,40)
40 FORMAT('RNR ',5X,' SNR ',5X,' A ')
DO 70 I=1,NN1
A(I)=0.0DO
RNR(I)=0
SNR(I)=0
70 CONTINUE
DO 120 I=1,2
Z=IZ
DO 30 J=1,Z
A(J)=B(J)
RNR(J)=RBR(J)
SNR(J)=SBR(J)
30 CONTINUE
DO 90 J=1,Z
IF(RNR(J).NE.SNR(J)) GOTO 90
A(J)=A(J)-V(I)
90 CONTINUE
CALL DETER3(N,Z,A,SNR,NN,RNR,NN1,PIVOT,HA,IHA,AFLAG,1
IFLAG,IFAIL,DET)
Z=IZ
IF(DET.NE.0.0DO) GOTO 100
LAMBDA=V(I)
GOTO 190
100 F(I)=DET
IF(II.LT.2) GOTO 120
DEN=1.0DO
DO 110 IP=1,II-1
DEN=DEN*(V(I)-EIGEN(IP))
110 CONTINUE
F(I)=DET/DEN
120 CONTINUE
130 LAMBDA=V(2)-(V(2)-V(1))*F(2)/(F(2)-F(I))
IF(DABS(LAMBDA-V(2)).LT.0.0000001) GOTO 190
F(1)=F(2)
V(1)=V(2)
V(2)=LAMBDA
DO 140 I=1,NN1
A(I)=0.0DO
SNR(I)=0
RNR(I)=0
140 CONTINUE
Z=IZ
DO 150 I=1,Z
A(I)=B(I)
SNR(I)=SBR(I)
RNR(I)=RBR(I)
150 CONTINUE
DO 160 I=1,Z
IF(RNR(I).NE.SNR(I)) GOTO 160
A(I)=A(I)-V(2)
160 CONTINUE
CALL DETER3(N,Z,A,SNR,NN,RNR,NN1,PIVOT,HA,IHA,AFLAG,1
IFLAG,IFAIL,DET)
Z=IZ
F(2)=DET
LAMBDA=DET
IF(DET.EQ.0.0DO) GOTO 190
IF(II.LT.2) GOTO 180
DEN=1.0DO
DO 170 IP=1,II-1
DEN=DEN*(V(2)-EIGEN(IP))
170 CONTINUE
F(2)=DET/DEN
180 GOTO 120
190 WRITE(6,200) LAMBDA
200 FORMAT('EIGEN VALUE=',E25.15)
EIGEN(II)=LAMBDA
IFAIL=1
IF(MFLAG.EQ.1) GOTO 390
DO 210 JJ=1,N
BB(JJ)=C(JJ)
210 CONTINUE
CALL Y12MDF(N,A,NN,BB,PIVOT,SNR,HA,IHA,IFLAG,IFAIL)
XMAX=0.0DO
DO 220 JJ=1,N
IF(DABS(BB(JJ)).LT.DABS(XMAX)) GOTO 220
XMAX=BB(JJ)
220 CONTINUE
DO 230 JJ=1,N
BB(JJ)=BB(JJ)/XMAX
230 CONTINUE
IF(MFLAG.EQ.3) GO TO 255
WRITE(6,240)
240 FORMAT('THE EIGENVECTOR CORRESPONDING TO THE ABOVE EIGENVALUE')
WRITE(6,250) (BB(JJ),JJ=1,N)
250 FORMAT(E30.20)
255 CONTINUE
DO 260 IS=1,NN1
A(IS)=0.0DO
RNR(IS)=0
SNR(IS)=0
260 CONTINUE
DO 270 IS=1,N
A(IS)=B(IS)
RNR(IS)=RBR(IS)
SNR(IS)=SBR(IS)
270 CONTINUE
IF(MFLAG.EQ.2) GOTO 390
DO 290 JS=1,N
S=0.0DO
DO 280 IS=1,N
IF(SNR(IS).EQ.JS) S=S+A(IS)*BB(RNR(IS))
280 CONTINUE
W(JS)=S
290 CONTINUE
S1=0.0DO
DO 300 IS=1,N
S1=S1+W(IS)*BB(IS)
300 CONTINUE
S2=0.0DO
DO 310 IS=1,N
S2=S2+BB(IS)*BB(IS)
310 CONTINUE
LAMBDA=S1/S2
WRITE(6,320) LAMBDA
320 FORMAT('THE IMPROVED EIGEN VALUE=',E30.20)
IF(MFLAG.EQ.3) GO TO 390
DO 330 IS=1,Z
IF(RNR(IS).NE.SNR(IS)) GO TO 330
A(IS)=A(IS)-LAMBDA
330 CONTINUE
CALL DETER(N,Z,A,SNR,NN,RNR,NN1,PIVOT,HA,IHA,AFLAG,IFLAG,1)
IFAIL,DET)
Z=IZ
DO 340 IS=1,N
BB(IS)=C(IS)
340 CONTINUE
CALL Y12DIF(N,A,NN,BB,PIVOT,SNR,HA,IHA,IFLAG,IFAIL)
XMAX=0.0D0
DO 350 JJ=1,N
IF(DABS(BB(JJ)).LT.DABS(XMAX)) GO TO 350
XMAX=BB(JJ)
350 CONTINUE
DO 360 JJ=1,N
BB(JJ)=BB(JJ)/XMAX
360 CONTINUE
WRITE(6,370)
370 FORMAT('THE IMPROVED EIGEN VECTOR BY INVERSE ITERATION')
WRITE(6,380)
380 FORMAT('**********************************************',//)
WRITE(6,250) (BB(JJ),JJ=1,N)
390 CONTINUE
RETURN
SUBROUTINE Y12MBF(N, Z, A, SNR, NN, RNR, NN1, HA, IHA, AFLAG, l IFAIL)

********************************************************

* THE NON-ZERO ELEMENTS OF THE MATRIX (A-LAMBDA*I) *
* GIVEN BY THE USER IN THE MULLER OR SECANT SUBROUTINES *
* ARE NOW ORDERED BY ROWS IN ARRAY A AND THEIR COLUMN *
* NUMBERS ARE STORED AT THE SAME POSITIONS IN ARRAY SNR.*
* THESE TWO ARRAYS FORM THE ROW ORDERED LIST.ANOTHER ST_*
* RUCTURE WHERE THE NON-ZERO ELEMENTS ARE ORDERED IN *
* BY COLUMNS IS PREPARED,HOWEVER,ONLY THE ROW NUMBERS OF* *
*THE NON-ZERO ELEMENTS ORDERED IN THIS WAY ARE STORED *
*IN ARRAY RNR.INFORMATION ABOUT THE ROW STARTS,ROW ENDS*
* ,THE COLUMN STARTS AND THE COLUMN ENDS IS STORED IN *
*ARRAY HA BY THIS SUBROUTINE.Y12MDF IS USED IN THE ROU.*
*TIME DETER. *

********************************************************

INITIALISATION OF THE PARAMETERS

****************************************************

THE INITIALISATIONS OF ALL THE PARAMETERS ARE DESCRIBED
IN THE SUBROUTINES MULLER AND DETER.

DOUBLE PRECISION A(NN), AFLAG(S)
INTEGER SNR(NN), RNR(NN1), HA(IHA+11), IFLAG(10)
MODE=IFLAG(4)
IF(NN,GE.2*Z)GO TO 1
IFLAG(1)=3
GO TO 22
1 IF(NN1,GE.Z)GO TO 2
IFLAG(1)=4
GO TO 22
2 G1=0.0000
IFLAG(1)=0
DO 10 I=1,N
   HA(I,2)=0
   HA(I,3)=0
10 HA(I,6)=0

FIND THE NUMBER OF THE NON-ZERO ELEMENTS IN EACH ROW AND COLUMNI MOVE
THE NON-ZERO ELEMENTS IN THE END OF THE ARRAYS A AND SNRFIND THE
LARGEST NON-ZERO ELEMENT IN A(IN ABSOLUTE VALUE).

DO 20 I=1,Z
   T=DABS(A(I))
   L3=RNR(I)
   L4=SNR(I)
   HA(L3,3)=HA(L3,3)+1
   HA(L4,6)=HA(L4,6)+1
20 CONTINUE
IF(T.GT.GT1)GT1=T
A(Z+I)=A(I)
20 SNR(Z+I)=SNR(I)
C
C STORE THE INFORMATION OF THE ROW STARTS(IN HA(I,1)) AND OF THE COLUMN
C STARTS(IN HA(I,4)).
C
L1=1
L2=1
DO 40 I=1,N
L3=HA(I,3)
L4=HA(I,6)
IF(L3.GT.0)GO TO 21
IFLAG(I)=5
GO TO 22
21 IF(L4.GT.0)GO TO 23
IFLAG(I)=6
GO TO 22
23 IF(MODE.EQ.2)GO TO 30
HA(I,9)=L3
HA(I,10)=L4
HA(I,11)=0
HA(L3,2)=HA(L3,2)+1
HA(I,5)=L3
30 HA(I,1)=L1
HA(I,4)=L2
L1=L1+L3
L2=L2+L4
HA(I,3)=0
40 HA(I,6)=0
C
C STORE THE NON-ZERO ELEMENTS OF MATRIX A (ORDERED IN ROWS) IN THE
C FIRST Z LOCATIONS OF THE ARRAY A. DO THE SAME FOR THEIR COLUMN NUMBERS
C
DO 50 I=1,Z
L1=Z+I
L3=SNR(I)
L2=HA(L3,1)+HA(L3,3)
A(L2)=A(L1)
SNR(L2)=SNR(L1)
50 HA(L3,3)=HA(L3,3)+1
C
C STORE THE ROW NUMBERS OF THE NON-ZERO ELEMENTS ORDERED BY COLUMNS IN
C THE FIRST Z LOCATIONS OF THE ARRAY RNR. STORE INFORMATION ABOUT ROW
C ENDS (IN HA(I,3)).
C
L4=1
DO 70 I=1,N
IF(MODE.EQ.2)GO TO 60
IF(HA(I,2).EQ.0)GO TO 60
HA(I,11)=L4
L4=L4+HA(I,2)
HA(I,2)=HA(I,11)
60 HA(I,3)=HA(I,1)+HA(I,3)-1
L1=HA(I,1)
L2=HA(I,3)
DO 70 J=L1,L2
   L3=SNR(J)
   INDEX=HA(L3,4)+HA(L3,6)
   RNR(INDEX)=1
70  HA(L3,6)=HA(L3,6)+1
   DO 90 I=1,N
   IF(MODE.EQ.2)GO TO 80
   L3=HA(I,5)
   L5=HA(L3,2)
   HA(L5,8)=I
   HA(I,7)=L5
   HA(L3,2)=HA(L3,2)+1
80  CONTINUE
90  HA(I,6)=HA(I,4)+HA(I,6)-1
   SNR(Z+1)=0
   RNR(Z+1)=0
   AFLAG(6)=GT1
   IFLAG(6)=0
   IFLAG(7)=0
   IFLAG(8)=Z
   22 FJL=IFLAG(1)+2
      IFAIL=0
      IF(FJL.NE.2) WRITE(6,100) IFAIL
   100  FORMAT('ERROR IFAIL=',I5)
      IF(FJL.EQ.2) IFLAG(1)=-1
   RETURN
END
SUBROUTINE Y12MJF(N,Z,A,SNR,NN,RNR,NN1,PIVOT,HA,IHA,
1 AFLAG,IFLAG,IFAIL,MARK1)

DOUBLE PRECISION A(NN),PIVOT(N),AFLAG(S)
INTEGER SNR(NN),RNR(NN1),HA(IHA,11),IFLAG(10)
INTEGER R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,SLUT,RR1,RR2,RR3,RR4,
1C1,C2,RROW,RCOLL,RPIVOT,CR1,CR2,CR3,CR4,RR,ZZ

**********************************************************************
* THIS SUBROUTINE FACTORIZES THE MATRIX (A-LAMBDA*I)*
* INTO TWO TRIANGULAR MATRICES AND FINDS THE PIVOTAL ELEMENTS REQUIRED FOR THE EVALUATION OF THE DETERMINANT *
* OF (A-LAMBDA*I) AND PASSES THEM THROUGH THE DETER SUBROUTINE TOGETHER WITH THE TOTAL NUMBER OF INTERCHANGES*
* MADE DURING THE FACTORISATION PROCESS WHICH IS STORED IN MARK1.Y12MJF ALSO PROVIDES THE INFORMATION REQUIRED*
* FOR THE BACK SUBSTITUTION PROCESS IN ORDER TO BE ABLE TO EVALUATE THE EIGENVECTORS. *
**********************************************************************

INITIALISATION OF THE PARAMETERS
======================================

THE INITIALISATIONS OF THE PARAMETERS ARE DESCRIBED IN THE SUBROUTINES DETER AND MULLER.

INFORMATION WHICH IS NECESSARY TO BEGIN THE ELIMINATION IS STORED.

IF(IFLAG(1).NE.-1)GO TO 1110
N8=N+1
N7=N-1
GRMIN=AFLAG(4)*AFLAG(6)
ZZ=Z

USE THE INFORMATION ABOUT FILL-INS IF IT IS POSSIBLE.

NR=N*N
IF(IFLAG(4).NE.2)GO TO 100
IF(IFLAG(10).GT.NN)GO TO 50
L1=IFLAG(10)
L5=L1+1
IF(L5.LE.NN)SNR(L5)=0
DO 40 I=1,N
L=N8-I
L2=HA(L,3)+1
L3=L2-HA(L,1)
DO 10 J=1,L3
SNR(L5-J)=SNR(L2-J)

40 CONTINUE
10 CONTINUE
10 A(L5-J)=A(L2-J)
   HA(L+3)=L1
   HA(L+1)=L5-L3
   L6=L1-L3
   L5=L5-HA(L,9)
   IF(L5.GT.L6)GO TO 30
   DO 20 J=L5,L6
20 SNR(J)=0
30 CONTINUE
40 L1=L5-1
50 IF(IFLAG(9).GT.NN1)GO TO 100
   L2=IFLAG(9)
   L5=L2+1
   IF(L5.LE.NN1)RNR(L5)=0
   DO 90 I=1,N
      L=N8-I
      L1=HA(L,6)+1
      L4=L1-HA(L,4)
      DO 60 J=1,L4
60 RNR(L5-J)=RNR(L1-J)
   HA(L,4)=L5-L4
   HA(L,6)=L2
   L6=L2-L4
   L5=L5-HA(L,10)
   IF(L5.GT.L6)GO TO 80
   DO 70 J=L5,L6
70 RNR(J)=0
80 CONTINUE
90 L2=L5-1
100 R4=HA(N,3)
   R5=HA(N,6)
   AFLAG(7)=AFLAG(6)
   AFLAG(8)=AFLAG(6)
   DO 110 I=1,N
      PIVOT(I)=0.000
      HA(I,2)=HA(I,1)
      HA(I,5)=HA(I,4)
      INDEX=HA(N,8)
      SLUT=HA(INDEX,3)-HA(INDEX,2)+1
110 CONTINUE
C START OF GAUSSIAN ELIMINATION.
C
MARK1=0
DO 950 I=1,N7
RR3=HA(I,2)
RR4=HA(I,3)
C1=HA(I,4)
CR4=HA(I,6)
IF(IFLAG(3).EQ.0)GO TO 350
IF(IFLAG(4).NE.2)GO TO 120
RRW=HA(I,7)
RCOLL=HA(I,8)
GO TO 220
120 L4=HA(I,8)
   IF(IFLAG(3).EQ.1)GO TO 130
   RROW=L4
   DO 130 J=1,N7
      RROW=RRW-J+1
      RRW=RROW
      GO TO 120
130 RROW=RRW
950 CONTINUE
C
REMOVE THE PIVOT ROW OF THE LIST WHERE THE ROWS ARE ORDERED BY INCREASING NUMBERS OF NON-ZERO ELEMENTS.

L1=0
L=1
L2=HA(L4,3)-HA(L4,2)+1
180 L=L+1
IF(L2.GT.L1)HA(L2,11)=L
IF(L.GT.N)GO TO 190
L5=HA(L,8)
L3=HA(L5,3)-HA(L5,2)+1
IF(RPIVOT.LT.L)GO TO 190
HA(L4,7)=L
HA(L,8)=L4
L4=L5
L1=L2
L2 = L3
L3 = N8
GO TO 180

190 IF(L2.EQ.L1)GO TO 200
IF(L3.EQ.L2)GO TO 200
HA(L2+11) = 0

200 L5 = HA(I, 7)
IF(RROW.EQ.I)GO TO 210
HA(L5+8) = RROW
HA(RROW, 7) = L5

210 HA(I, 7) = RROW
HA(I, 8) = RCOLL

C C ROW INTERCHANGES.

220 IF(RROW.EQ.I)GO TO 290
MARK1 = MARK1 + 1
DO 250 J = R3, R4
L1 = SNR(J)
R = HA(L1, 5) - 1
R10 = HA(L1, 6)

240 R = R + 1
IF(RNR(R).NE.I)GO TO 240
RNR(R) = RNR(R10)

250 RNR(R10) = RROW
R3 = HA(RROW, 2)
R4 = HA(RROW, 3)
DO 270 J = R3, R4
L1 = SNR(J)
R = HA(L1, 5) - 1

260 R = R + 1
IF(RNR(R).NE.RROW)GO TO 260

270 RNR(R) = I
DO 280 J = 1, 3
R3 = HA(RROW, J)
HA(RROW, J) = HA(I, J)

280 HA(I, J) = R3

C C COLUMN INTERCHANGES.

290 IF(RCOLL.EQ.I)GO TO 350
MARK1 = MARK1 + 1
DO 310 J = C1, C4
L1 = RNR(J)
R = HA(L1, 2) - 1
R10 = HA(L1, 3)

300 R = R + 1
IF(SNR(R).NE.I)GO TO 300
T = A(R10)
A(R10) = A(R)
A(R) = T
SNR(R) = SNR(R10)

310 SNR(R10) = RCOLL
C1 = HA(RCOLL, 4)
C4 = HA(RCOLL, 6)
DO 330 J = C1, C4
L1=RNR(J)
R=HA(L1,2)-1
320 R=R+1
    IF(SNR(R),NE.,RCOLL)GO TO 320
330 SNR(R)=I
    DO 340 J=4,6
    HA(RCOLL,J)=HA(I,J)
    HA(I,J)=R3
340 HA(I,J)=R3
C END OF THE INTERCHANGES.
C THE ROW ORDERED LIST AND THE COLUMN ORDERED LIST ARE PREPARED TO
C BEGIN STEP 1 OF THE ELIMINATION.

350 R9=RR4-RR3
    DO 360 RR=RR3,RR4
        IF(SNR(RR),EQ.,I)GO TO 370
360 CONTINUE
    IFLAG(1)=7
    GO TO 1110
370 V=A(RR)
    PIVOT(I)=V
    TD=DABS(V)
        IF(TD,LT.,AFLAG(8))AFLAG(8)=TD
        IF(TD,GE.,GRMIN)GO TO 380
    IFLAG(1)=1
    GO TO 1110
380 R2=HA(I,1)
    A(RR)=A(RR3)
    SNR(RR)=SNR(RR3)
    A(RR3)=A(R2)
    SNR(RR3)=SNR(R2)
    SNR(R2)=0
    Z=Z-1
    RR3=RR3+1
    HA(I,2)=RR3
    HA(I,1)=R2+1
    CR3=HA(I,5)
        IF(R9,LE.,0)GO TO 431
    DO 430 J=RR3,RR4
        INDEX=SNR(J)
430 PIVOT(INDEX)=A(J)
431 R7=CR4-CR3+1
    DO 880 K=1,R7
    R1=RNR(CR3-1+K)
        IF(R1,EQ.,I)GO TO 870
    I1=HA(R1,1)
    RR1=HA(R1,2)
    RR2=HA(R1,3)
    L2=RR2-RR1+1
    L=RR1-1
390 L=L+1
        IF(SNR(L),NE.,I)GO TO 390
    T=A(L)/V
        IF(IFLAG(5),EQ.,2)GO TO 400
    A(L)=A(I1)


```
SNR(L)=SNR(I1)
SNR(I1)=0
I1=I1+1
HA(R1,1)=I1
Z=Z-1
GO TO 410
400 A(L)=A(RR1)
A(RR1)=T
R3=SNR(RR1)
SNR(RR1)=SNR(L)
SNR(L)=R3
410 RR1=RR1+1
HA(R1,2)=RR1
IF(R9.LE.0) GO TO 760
R=RR1
IF(R.GT.RR2) GO TO 470
DO 460 L=R,RR2
L1=SNR(L)
TD=PIVOT(L1)
IF(TD.EQ.0.0D0) GO TO 450
PIVOT(L1)=0.0D0
TD=A(L)-TD*T
A(L)=TD
TD1=DABS(TD)
IF(TD1.GT.AFLAG(7)) AFLAG(7)=TD1
IF(TD1.GT.AFLAG(2)) GO TO 450
C TOO SMALL ELEMENT IS CREATED. REMOVE IT FROM THE LISTS.
C Z=Z-1
A(L)=A(RR1)
SNR(L)=SNR(RR1)
A(RR1)=A(I1)
SNR(RR1)=SNR(I1)
SNR(I1)=0
RR1=RR1+1
I1=I1+1
HA(R1,2)=RR1
HA(R1,1)=I1
R3=HA(L1,5)
R2=R3-1
L4=HA(L1,4)
L5=RNR(L4)
L6=RNR(R3)
440 R2=R2+1
IF(RNR(R2).NE.R1) GO TO 440
RNR(R2)=L6
RNR(R3)=L5
RNR(L4)=0
HA(L1,5)=R3+1
HA(L1,4)=L4+1
450 CONTINUE
460 CONTINUE
470 CONTINUE
DO 750 J=I,R9
R=RR3-1+J
```
Y12MJF.F PAGE 7

R2=SNR(R)
TOL2=PIVOT(R2)
PIVOT(R2)=A(R)
IF(TOL2.EQ.0.0D0)GO TO 740
TOL3=-TOL2*T
TOL1=DABS(TOL3)
IF(TOL1.LT.AFLAG(2))GO TO 740
C2=HA(R2,4)
CR2=HA(R2,6)
CR1=HA(R2,5)
LFR=RR2-I1+2
LFC=CR2-C2+2
IF(IFLAG(4).NE.1)GO TO 480
IF(LFR.GT.HA(R1,9))HA(R1,9)=LFR
IF(LFC.GT.HA(R2,10))HA(R2,10)=LFC
480 IF(I1.EQ.1)GO TO 490
490 IF(RR2.EQ.NN)GO TO 500
500 R10=NN-LFR
C
C COLLECTION IN ROW ORDERED LIST.
C
IF(R10.GE.R4)GO TO 560
IFLAG(6)=IFLAG(6)+1
DO 520 JJ=1,N
L1=HA(JJ,3)
IF(L1.LT.HA(JJ,1))GO TO 510
HA(JJ,3)=SNRCL1
SNRCL1>=-JJ
510 CONTINUE
520 CONTINUE
L3=0
L4=1
DO 550 JJ=1,R4
IF(SNR(JJ).EQ.0)GO TO 540
L3=L3+1
IF(SNR(JJ).GT.0)GO TO 530
L5=-SNR(JJ)
SNR(JJ)=HA(L5,3)
HA(L5,3)=L3
L6=L4+HA(L5,2)-HA(L5,1)
HA(L5,2)=L6
HA(L5,1)=L4
L4=L3+1
530 A(L3)=A(JJ)
SNR(L3)=SNR(JJ)
540 CONTINUE
550 CONTINUE
R4=L3
SNR(L3+1)=0
RR3=HA(1,2)
RR4=HA(1,3)
I1=HA(R1,1)
RR1=HA(R1,2)
R=RR3-1+J
IF(R10.GE.R4)GO TO 560
IFLAG(1)=3
GO TO 1110

C FILL-IN TAKES PLACE IN THE ROW ORDERED LIST.

560 R8=LFR-1
RR2=R4+LFR
L3=I1-1
DO 570 LL=1,R8
L4=R4+LL
L5=L3+LL
A(L4)=A(L5)
SNR(L4)=SNR(L5)
570 SNR(L5)=0
RR1=R4+RR1-I1+1
HA(R1,3)=RR2
HA(R1,2)=RR1
I1=R4+1
HA(R1,1)=I1
L1=RR2
GO TO 590
580 RR2=RR2+1
HA(R1,3)=RR2
L1=RR2
IF(RR2.LE.R4)GO TO 610
590 R4=RR2
IF(R4.LT.NN)SNR(R4+1)=0
GO TO 610
600 RR1=RR1-1
I1=I1-1
HA(R1,1)=I1
HA(R1,2)=RR1
L1=RR1
SNR(I1)=SNR(L1)
A(I1)=A(L1)
610 A(L1)=TOL3
SNR(L1)=SNR(R)
TD=DABS(A(L1))
IF(TD.GT.AFLAG(7))AFLAG(7)=TD
Z=Z+1
IF(IFLAG(8).LT.Z) IFLAG(8)=Z
IF(C2.EQ.1)GO TO 620
IF(RNR(C2-1).EQ.0)GO TO 720
620 IF(CR2.EQ.NN1)GO TO 630
IF(RNR(CR2+1).EQ.0)GO TO 700
630 R10=NN1-LFC

C COLLECTION IN COLUMN ORDERED LIST.

C IF(R10.GE.R5)GO TO 680
IFLAG(7)=IFLAG(7)+1
DO 640 JJ=I,N
L1=HA(JJ+6)
HA(JJ+6)=RNR(L1)
640 RNR(L1)=-JJ
L3=0
L4=1
DO 670 JJ=1,R5
IF(RNR(JJ).EQ.0)GO TO 660
L3=L3+1
IF(RNR(JJ).GT.0)GO TO 650
L5=-RNR(JJ)
RNR(JJ)=HA(L5,6)
HA(L5,6)=L3
L6=L4+HA(L5,5)-HA(L5,4)
HA(L5,4)=L6
HA(L5,5)=L4
L4=L3+1
650 RNR(L3)=RNR(JJ)
660 CONTINUE
670 CONTINUE
R5=L3
RNR(R5+1)=0
C2=HA(R2,4)
CR3=HA(I,5)
CR4=HA(I,6)
CR1=HA(R2,5)
IF(R10.GE.R5)GO TO 680
IFLAG(1)=4
GO TO 1110
C FILL-IN TAKES PLACE IN THE COLUMN ORDERED LIST.
C
680 R8=LFC-1
CR2=R5+LFC
L3=C2-1
DO 690 L=1,R8
L4=R5+L
L5=L3+L
RNR(L4)=RNR(L5)
690 RNR(L5)=0
CR1=R5+CR1-C2+1
C2=R5+1
HA(R2,6)=CR2
HA(R2,4)=C2
HA(R2,5)=CR1
R=CR2
GO TO 710
700 CR2=CR2+1
HA(R2,6)=CR2
R=CR2
IF(CR2.LE.R5)GO TO 730
710 R5=CR2
IF(R5.LT.NN1)RNR(R5+1)=0
GO TO 730
720 CR1=CR1-1
C2=C2-1
HA(R2,4)=C2
HA(R2,5)=CR1
R=CR1
RNR(C2)=RNR(R)
C
C UPDATE THE INFORMATION IN THE LIST WHERE THE ROWS ARE ORDERED BY
C INCREASING NUMBERS OF THE NON-ZERO ELEMENTS.
C
760 IF(IFLAG(4).EQ.2)GO TO 870
   IF(IFLAG(3).EQ.0)GO TO 870
   L1=RR2-RR1+1
   IF(L1.EQ.L2)GO TO 870
   L6=HA(R1,7)
   L4=HA(L2,11)
   IF(L1.GT.L2)GO TO 820
   IF(L6.GT.L4)GO TO 780
   IF(L4.EQ.N)GO TO 770
   L=HA(L4+1,8)
   L5=HA(L,3)-HA(L,2)+1
   IF(L5.EQ.L2)GO TO 790
770 HA(L2,11)=0
   GO TO 800
780 L5=HA(L4,8)
   L3=HA(L6,8)
   HA(L4,8)=L3
   HA(L6,8)=L5
   HA(L5,7)=L6
   HA(L3,7)=L4
   L6=L4
790 HA(L2,11)=L4+1
800 IF(L4.EQ.I+1)GO TO 810
   L=HA(L6-1,8)
   L2=HA(L,3)-HA(L,2)+1
   L4=HA(L2,11)
   IF(L1.LT.L2)GO TO 780
810 IF(L1.NE.L2)HA(L1,11)=L6
   GO TO 870
820 IF(L6.GT.L4)GO TO 840
   IF(L4.EQ.N)GO TO 830
   L=HA(L4+1,8)
   L5=HA(L,3)-HA(L,2)+1
   IF(L5.EQ.L2)GO TO 840
830 HA(L2,11)=0
840 L2=L2+1
   IF(L2.LE.SLUT)GO TO 850
   L3=N
   SLUT=L1
   L2=L1
   GO TO 860
850 L3=HA(L2,11)-1
   IF(L3.EQ.-1)GO TO 840
   IF(L2.GT.L1)L2=L1
860 HA(L2,11)=L3
   L4=HA(L3,9)
L7=HA(L6,8)
HA(L3,8)=L7
HA(L6,8)=L4
HA(L7,7)=L3
HA(L4,7)=L6
L6=L3
IF(L2.LT.L1)GO TO 840
870 CONTINUE
880 CONTINUE
IF(R9.LE.0)GO TO 882
DO 881 J=RR3,RR4
INDEX=SNR(J)
881 PIVOT(INDEX)=0.000
882 CONTINUE
CR3=HA(I,4)
DO 890 J=CR3,CR4
890 RNR(J)=0
IF(R9.LE.0)GO TO 930
L2=HA(I,2)-1
DO 920 LL=1,R9
R=SNR(L2+LL)
R1=HA(R,5)
R2=HA(R,6)
IF(R2.GT.R1)GO TO 900
IFLAG(1)=6
GO TO 1110
900 HA(R,5)=R1+1
R3=R1-1
910 R3=R3+1
IF(RNR(R3).NE.1)GO TO 910
RNR(R3)=RNR(R1)
920 RNR(R1)=I
930 AFLAG(5)=AFLAG(7)/AFLAG(6)
IF(AFLAG(5).LT.AFLAG(3))GO TO 940
IFLAG(1)=2
GO TO 1110
940 CONTINUE
950 CONTINUE
C
C PREPARATION TO BEGIN THE BACK SUBSTITUTION.
C
INDEX=HA(N,2)
PIVOT(N)=A(INDEX)
A(INDEX)=0.000
TD=DABS(PIVOT(N))
IF(TD.GT.AFLAG(7))AFLAG(7)=TD
IF(TD.LT.AFLAG(8))AFLAG(8)=TD
IF(TD.GT.GRMIN)GO TO 960
IFLAG(1)=1
GO TO 1110
960 IF(IFLAG(4).NE.1)GO TO 1060
IFLAG(10)=HA(N,9)
IFLAG(9)=HA(N,10)
DO 990 I=1,N7
R1=N-I
IFLAG(10)=IFLAG(10)+HA(R1,9)
990 CONTINUE
IFLAG(9) = IFLAG(9) + HA(R1, 10)
IF (IFLAG(3).EQ.0) GO TO 980
DO 970 J = 9, 10
R2 = HA(R1, J - 2)
R6 = HA(R2, J)
HA(R2, J) = HA(R1, J)
970 HA(R1, J) = R6
980 CONTINUE
990 CONTINUE
CONTINUE
AFLAG(5) = AFLAG(7) / AFLAG(6)
1110 Z = Z
IF (IFLAG(1).NE.-1) GO TO 1120
IFAIL = 0
IFLAG(1) = -2
RETURN
1120 WRITE(6, 1130) IFAIL
1130 FORMAT('ERROR IFAIL= ', I5)
RETURN
END
SUBROUTINE Y12MDF(N,AN,B,PIVOT,SNR,HA,IHA,IFLAG,IFAIL)

DOUBLE PRECISION A(AN), PIVOT(N), B(N)
INTEGER SNR(AN), HA(IHA,11), IFLAG(10)

***********************************************************
** THIS SUBROUTINE EVALUATES THE EIGENVECTORS OF A 
** MATRIX WHICH HAS ALREADY BEEN FACTORISED BY THE ROUTINE 
** Y12MJF. THE SUBROUTINE Y12MDF PERFORMS THE BACK SUBSTITUTION PROCESS AND DETERMINE THE REQUIRED EIGENVECTOR. 
***********************************************************

INITIALISATION OF THE PARAMETERS
====================================================================
THE INITIALISATIONS OF ALL THE PARAMETERS ARE DESCRIBED
IN THE SUBROUTINES DETER AND MULLER.

IF(IFLAG(1).NE.-2)GO TO 1110
MODE=IFLAG(4)
IPIV=IFLAG(3)
NS=N+1
N7=N-1
STATE=IFLAG(5)

SOLVE THE SYSTEM WITH LOWER TRIANGULAR MATRIX L (IF THE
LU-FACTORIZATION IS AVAILABLE).

IF(STATE.NE.3)GO TO 1051
IF(IPIV.EQ.0)GO TO 1020
DO 1010 I=1,N7
L1=HA(I,7)
T=B(L1)
B(L1)=B(I)
B(I)=T
1010 CONTINUE

1020 CONTINUE
DO 1050 I=1,N
RR1=HA(I,1)
RR2=HA(I,2)-1
IF(RR1.GT.RR2)GO TO 1040
DO 1030 J=RR1,RR2
L1=SNR(J)
1030 B(I)=B(I)-A(J)*B(L1)
1040 CONTINUE
1050 CONTINUE

SOLVE THE SYSTEM WITH UPPER TRIANGULAR MATRIX.
C
1051 CONTINUE
   DO 1090 I=1,N
      R1=N8-I
      RR1=HA(R1,2)
      RR2=HA(R1,3)
      IF(RR2.LT.RR1) GO TO 1080
   DO 1070 J=RR1,RR2
      R2=SNR(J)
      1070 B(R1)=B(R1)-A(J)*B(R2)
   1080 CONTINUE
   B(R1)=B(R1)/PIVOT(R1)
   1090 CONTINUE
C IF INTERCHANGES WERE USED DURING THE ELIMINATION THEN A REORDERING IN C LUTION VECTOR IS MADE.
   IF(IPIV.EQ.0)GO TO 1110
   DO 1100 I=1,N7
      R1=N-I
      R2=HA(R1,8)
      T=B(R2)
      B(R2)=B(R1)
   1100 B(R1)=T
   1110 IF(IFLAG(1).NE.-2)GO TO 1120
   IFAIL=0
   RETURN
   1120 WRITE(6,1130) IFAIL
   1130 FORMAT('ERROR IFAIL=',I5)
   RETURN
END
APPENDIX II

FORTRAN PROGRAMS ON ITERATIVE METHODS

FOR THE SOLUTION OF EIGENSYSTEMS
subroutine precond(m1,m2,m3,iflag,ia,ja,ia,b,b,ib,c,ic,l,l,jl,jl,
1 il,ju,il,uu,il,dd,diag,jdd,idd,y1,y2,z,xw,lambda)

dimension a(m1),ja(m1),ia(m2),b(m3),jb(m3),ib(m2),c(m3),ic(m3),
1 ic(m2),il(m3),j(m2),il(m2),uu(m3),iu(m2),il(m3),iu(m2),il(m3),uu(m3),
2 di(m2),jdd(m2),idd(m2),y1(m2),y2(m2),z(m2),x(m2),diag(m2),
3 y(m2)

real il,l,lambda

*************************************************************************
* THIS SUBROUTINE EVALUATES THE EXTEREM EIGENVALUES AND *
* THEIR CORRESPONDING EIGENVECTORS OF A REAL SYMMETRIC MATRIX *
* REPRESENTED HERE BY (a,ja,ia) ACCORDING TO THE PRE-CONDIT-** *
* METHOD DESCRIBED IN (V.A). *
*************************************************************************

INITIALISATION OF THE PARAMETERS
=============================================

m1 - DENOTES THE NUMBER OF THE NON-ZERO ELEMENTS IN THE COEF-
FICIENT MATRIX A.

m2 - IS EQUAL TO (n+1) WHERE n DENOTES THE SIZE OF THE MATRIX A.

m3 - IS EQUAL TO (m1/2).

iflag - THE USE OF THIS INTEGER VARIABLE IS DESCRIBED AS FOLLOWS;

iflag=1 - DENOTES THAT ONLY THE EIGENVALUE IS REQUIRED AND THE
SUBROUTINE WILL PRINT OUT THE COMPUTED VALUE OF THE
THE EIGENVALUE TOGETHER WITH THE CORRESPONDING VECTOR
WHICH IS A GOOD APPROXIMATION TO THE EIGENVECTOR.

iflag=0 - DENOTES THAT AFTER THE EVALUATION OF THE EIGENVALUE
THEN THE SUBROUTINE WILL STOP EVALUATING THE RAYLEIGH
QUOTIENT AND CONTINUES THE PROCESS TO OBTAIN AN EIGEN-
VECTO~ AS ACCURATE AS THE EIGENVALUE ITSELF.

a(m1) - THE ONE DIMENSIONAL REAL ARRAY a CONTAINS THE NON-ZERO
ELEMENTS OF THE MATRIX A IN THE ORDER THEY APPEAR ROW BY
POW AND MUST BE PROVIDED BY THE USER BEFORE CALLING THE
THE SUBROUTINE.UNCHANGED ON EXIT.

ja(m1) - THIS INTEGER ARRAY CONTAINS THE COLUMN NUMBERS OF THE NON-
ZERO ELEMENTS OF THE MATRIX A IN THE SAME ORDER AS THEY ARE
STORED IN THE ARRAY a. THIS VECTOR MUST BE INITIALISED BY
THE USER BEFORE CALLING THE SUBROUTINE.UNCHANGED ON EXIT.
ia(m2) - THE ONE DIMENSIONAL INTEGER ARRAY ia REPRESENTS THE NUMBER OF THE NON-ZERO ELEMENTS IN EACH ROW OF THE MATRIX A AS FOLLOWS:

WITH a(1) BEING INITIALISED TO 1 THEN ,a(2)-a(1) DENOTES THE NUMBER OF THE NON-ZERO ELEMENTS IN THE FIRST ROW OF A AND IN GENERAL, a(i)-a(i-1) GIVES THE NUMBER OF THE NON-ZERO ELEMENTS IN THE ROW (i-1). TO INITIALISE THIS VECTOR WE ADD THE NUMBER OF NON-ZERO ELEMENTS IN THE FIRST ROW TO THE VALUE OF a(1)=1 AND STORE IT IN a(2) AND THEN WE ADD THE NUMBER OF NON-ZERO ELEMENTS OF THE SECOND ROW TO a(2) AND STORE IT IN a(3) AND SO ON. THIS ARRAY MUST ALSO BE INITIALISED BY THE USER BEFORE CALLING THE SUBROUTINE.UNCHANGED ON EXIT.

w(m2) - THIS VECTOR IS USED FOR THE STORAGE OF THE COMPUTED EIGENVECTOR AND ON THE ENTRY TO THE SUBROUTINE MUST CONTAIN THE REQUIRED INITIAL VALUE OF THE EIGENVALUE.THIS VECTOR MUST BE INITIALISED BY THE USER.

lambda - THE INITIAL VALUE OF THE REQUIRED EIGENVALUE MUST BE PROVIDED AND STORED IN lambda BY THE USER. ON EXIT, THE COMPUTED EIGENVALUE WILL BE STORED IN THIS PARAMETER.

w - THIS IS THE PRECONDITIONED FACTOR WHICH MUST BE GIVEN BY THE USER.

(l,j1,i1) - THIS MATRIX IS USED FOR THE STORAGE OF THE LOWER TRIANGULAR COMPONENT OF THE MATRIX A THEREFORE THE SIZE OF l AND j1 IS m3=m1/2 AND THE SIZE OF i1 IS m2=n+1. THIS MATRIX IS GENERATED BY THE SUBROUTINE.

(u,j1,i1) - THIS MATRIX IS USED FOR THE STORAGE OF THE UPPER TRIANGULAR PART OF A AND IS OF THE SAME SIZE AS THE MATRIX (l,j1,i1). THIS MATRIX IS EVALUATED BY THE SUBROUTINE.

(b,jb,ib) - THIS MATRIX IS USED FOR THE STORAGE OF THE SUCCESSIVE EVALUATED MATRICES di-w1 AND HAS THE SAME SIZE AS THE MATRIX (l,j1,i1).

(c,jc,ic) - THE SUCCESSIVE EVALUATED MATRICES di-wu ARE STORED IN THIS MATRIX WHICH IS OF THE SAME SIZE AS (l,j1,i1).

OTHER ARRAYS - ALL THE OTHER ARRAYS USED IN THIS SUBROUTINE ARE USED AS WORKING SPACE. THE SIZE OF THESE ARRAYS ARE GIVEN IN THE DIMENSION STATEMENT ABOV.
n=m2-1
imark=0
eps=0.000001
10 format(f12.7)
20 format(i6)
do 40 i=1,n
do 30 k=ia(i),ia(i+1)-1
if(ia(k).ne.i) goto 30
diag(U=a(k)
goto 40
30 continue
40 continue
istep1=1
iter=0
write(6,50) w
50 format('THE VALUE OF P_C PARAMETER=',f12.7)
call fact(a,ja,ia,n,1,il,il,u,ju,iu)
do 60 k=1,1(n+1)-1
11(k)=(1-w)*1(k)
uu(k)=(1-w)*u(k)
l(k)=w*l(k)
u(k)=w*u(k)
60 continue
iter=0
istep2=istep1
eps=0.000001
write(6,70) istep1
70 format('ISTEP=',i6)
mark=0
write(6,80) lambda
80 format('THE INITIAL VALUE OF LAMBDA=',f12.7)
write(6,90)
90 format('***************************************************
')
do 390 i=1,1000
100 format(1./(d-lambda)
do 100 k=1,n
dd(k)=diag(k)-lambda
jdd(k)=k
100 continue
idd(n+1)=n+1
do 110 k=1,150
c(k)=0.
b(k)=0.
jb(k)=0.
jc(k)=0.
110 continue
do 120 k=1,n+1
ib(k) = 0
ic(k) = 0
continue
call spsub(dd, jdd, idd, n, n, 1, j1, il, n, n, b, jb, ib)

B = (D - LAMBDA) - w*L

call spsub(dd, jdd, idd, n, u, ju, iu, n, n, c, jc, ic)

C = (D - LAMBDA) - w*U

write(6, 130) omega
format(1, THE VALUE OF OMEGA :', f12.7, //)
imark = 1
do 330 j = 1, istep2
call vecmul(ll, j1, il, n, x, n, y1)

Y1 = (1-w) * L*X

call vecmul(uu, ju, iu, n, x, n, y2)

Y2 = (1-w) * U*X

call vecmul(u, ju, iu, n, x, n, z)

Z = U*X

do 140 k = 1, n
z(k) = z(k) / dd(k)
continue
call vecmul(l, j1, il, n, z, n, y)

Y = L*inverse(D) * U

do 150 k = 1, n
z(k) = y(k) + y1(k) + y2(k)

Z = L*inverse(D) * U*x + (1-w)* (L+U)*x

150 continue
xmax = 0.0
do 160 k = 1, n
if(abs(z(k)) .le. abs(xmax)) goto 160
xmax = z(k)

160 continue
do 170 k = 1, n
z(k) = z(k) / abs(xmax)

170 continue
call forsub(b, jb, ib, n, z)

Z = B*Z
do 180 k=1,n
   z(k)=z(k)*dd(k)
180 continue
   xmax=0.0
   do 190 k=1,n
      if(abs(z(k)).le.abs(xmax)) goto 190
      xmax=z(k)
190 continue
   do 200 k=1,n
      z(k)=z(k)/abs(xmax)
200 continue
   call baksu(c,jc,ic,n,z)
c
   Z=C*Z

dc
210 format(5(e25.15))
   xmax=0.0
   do 220 k=1,n
      if(abs(z(k)).lt.abs(xmax)) goto 220
      xmax=z(k)
220 continue
   if(xmax.eq.0.0) goto 250
230 format('XMAX=',f12.7)
c
   write(6,170) (z(k),k=1,n)
   do 240 k=1,n
      z(k)=z(k)/abs(xmax)
240 continue
250 do 260 k=1,n
   if(abs((x(k)-z(k))/(1.+abs(x(k)))),lt,eps) goto 260
   goto 310
260 continue
   iter=iter+j
   write(6,380) lambda
   write(6,300) (z(k),k=1,n)
c
280 format('*********** THE EIGENVECTOR ***********',/)
   write(6,300) (z(k),k=1,n)
290 format('ITER=',i6)
   write(6,270) iter
   write(6,300) lambda
300 format(5(f12.7,3x))
goto 410
310 do 320 k=1,n
   x(k)=z(k)
320 continue
330 continue
   iter=iter+istep2
   s1=0.0
   do 340 k=1,n
      s1=s1+x(k)**2
340 continue
   call vecmul(a,jia,ia,n,n,x,n,z)

THE NEW EVALUATION OF THE RAYLEIGH QUOTIENT.  

c
s=0.0
do 350 k=1,n
s=s+x(k)*z(k)
350 continue
if(mark.eq.1) goto 370
pp=s/s1
if(abs(pp-lambda).gt.eps) goto 370
write(6,360)
360 format('THE E/VALUE HAS CONVERGED TO THE REAL ANSWER',1, '
1 'AND THE E/VECTOR CORRESPONDING TO THIS VALUE IS GIVEN',1, '
2 'BELOW TOGETHER WITH THE NO. OF ITERATIONS SOFAR',1,1, '
3 'HOWEVER, THE PROCESS IS CONTINUED TO GET A BETTER ',1,1, '
4 'ESTIMATE OF THE E/VECTOR IF iflag=1.',1,1)
write(6,365) iter
365 format('THE NO. OF ITERATIONS SOFAR=',i6)
write(6,300) (x(k),k=1,n)
write(6,380) pp
mark=1
eps=eps/1.
istep2=5000
370 lambda=s/s1
write(6,380) lambda
380 format('LAMBDA=',f12.7,1)
if(iflag.eq.0) return
390 continue
write(6,400)
400 format('AFTER 5000 ITERATIONS DID NOT CONVERGED.1)
410 continue
return
end
subroutine pcmin(m1,m2,m3,i,flag,a,ia,ja,ia,ib,ib,c,ic,ic,jc,jc,l1,l1,
  1 i1,i1,ju,ju,iu,uu,ll,ll,dd,diag,jdd,idd,y1,y1,y2,z,z,w,lambda)

dimension a(m1),ja(m1),ia(m2),b(m3),ib(m2),c(m3),j(m3),
  1 ic(m2),i1(m3),j(m3),ii(m2),ju(m3),iu(m2),ll(m3),uu(m3),
  2 dd(m2),jdd(m2),idd(m2),y1(m2),y2(m2),z(m2),x(m2),diag(m2),
  3 y(m2)

real l1,l1,lambda

******************************************************************************
* THIS SUBROUTINE EVALUATES THE EXTEREM EIGENVALUES OF A *
* REAL SYMMETRIC MATRIX A BY THE MODIFIED PRE-CONDITIONED METHOD *
* DESCRIBED IN (VI.3). *
******************************************************************************

INITIALISATION OF THE PARAMETERS
========================================
ALL THE PARAMETERS ARE AS DEFINED IN THE prec:ond.f SUBROUTINE.

******************************************************************************
* THE INTERNAL SUBROUTINES USED IN THIS PROGRAM ARE: *
* FACT.F SPSUB.F SPADD.F VECMUL.F FORSUB.F BAKSUB.F *
* VECTOR.F *
******************************************************************************

n=m2-1
imark=0
t=eps=0.000001
10 format(f12.7)
20 format(i6)
do 40 i=1,n
do 30 k=ia(i),ia(i+1)-1
if(ja(k).ne.1) goto 30
diag(i)=a(k)
go to 40
30 continue
40 continue
istepl=l
iter=0
write(6,50) w
50 format('THE VALUE OF P_C PARAMETER=',f12.7/)
call fact(a,ia,ia,ia,n+1,l1,l1,ju,ju)
do 60 k=1,l(n+1)-1
ll(k)=(1-w)*a(k)
do 40
uu(k) = (1-w)*u(k)
l(k) = w*l(k)
u(k) = w*u(k)

continue
iter=0
istep2=istep1
eps=0.000001
write(6,70) istep1
70 format('ISTEP=',i6)
mark=0
write(6,80) lambda
80 format('THE INITIAL VALUE OF LAMBDA=',F12.7)
write(6,90)
90 format('***************************************************')
do 430 i=1,1000
omega=1.0/(d-lambda)
do 100 k=1,n
dd(k)=diag(k)-lambda
idd(k)=k
100 continue
idd(n+1)=n+1
do 110 k=1,150
c(k)=0.
b(k)=0.
jb(k)=0
jc(k)=0
110 continue
do 120 k=1,n+1
ib(k)=0
ic(k)=0
120 continue
call spsub(dd,jdd,idd,n,n+1,jl,il,n,n,b,jb,ib)

B=(D-LAMBDA)-w*L
call spsub(dd,jdd,idd,n,n+1,ju,iu,n,n,c,jc,ic)
C=(D-LAMBDA)-w*U

imark=1
do 360 j=1,istep2
if(mark.ne.1) goto 170
do 140 k=1,n
do 130 ik=ia(k),ia(k+1)-1
if(k.ne.ja(ik)) goto 130
a(ik)=a(ik)-lambda
goto 140
130 continue
140 continue
call vector(a,ja,ia,n,b,jb,ib,c,jc,ic,dd,x,z,iter)
do 160 k=1,n
do 150 ik=ia(k),ia(k+1)-1
if(k.ne.Ja(ik)) goto 150
a(ik)=a(ik)+lambda
goto 160
continue
160 continue
goto 300
call vecmul(ll,jl,il,n,n,x,n,y1)
c Y1=(1-w)*L*X
ccall vecmul(uu,ju,iu,n,n,x,n,y2)
c Y2=(1-w)*U*X
call vecmul(u,ju,iu,n,n,x,n,z)
c Z=U*X
do 180 k=1,n
z(k)=z(k)/dd(k)
continue
call vecmul(l,jl,il,n,n,z,n,y)
c Y=L*inverse(D)*U
do 190 k=1,n
z(k)=y(k)+y1(k)+y2(k)
continue
Z=L*inverse(D)*U*x+(1-w)*(L+U)*X
190 continue
xmax=0.0
do 200 k=1,n
if(abs(z(k)).le.abs(xmax)) goto 200
xmax=z(k)
200 continue
do 210 k=1,n
z(k)=z(k)/abs(xmax)
continue
call forsub(b,jb,ib,n,z)
c Z=B*Z
do 220 k=1,n
z(k)=z(k)*dd(k)
continue
xmax=0.0
do 230 k=1,n
if(abs(z(k)).le.abs(xmax)) goto 230
xmax=z(k)
continue
do 240 k=1,n
z(k)=z(k)/abs(xmax)
program pcmin

n parameters
n n: order of the matrix
n m: number of iterations
n tol: tolerance for convergence
n
n initialize variables
n j: iteration counter
n s: sum of squares of residuals
n x: vector of coefficients
n
n compute eigenvalue
n
j = 0
n
n repeat until convergence
n
n compute new vector
n
n compute new eigenvalue
n
n print results
n
n end program pcmin
write(6,400) iter
400 format('THE NO. OF ITERATIONS SOFAR=',i6)
write(6,400) (x(k),k=1,n)
write(6,420) pp
mark=1
eps=eps/1.
step2=5000
410 lambda=s/s1
write(6,420) lambda
420 format('LAMBDA=',f12.7,//)
if(flag.eq.0) return
430 continue
write(6,410)
440 format('AFTER 5000 ITERATIONS DID NOT CONVERGED.\n')
450 continue
return
end
subroutine shift(m1,m2,m3,iflag,a,ja,ia,b,jb,ib,c,jc,ic,l,jl,
1 il,u,ju,iu,diag,d,d2,idd,idd2,z,y,x,w,lambda)

real l,lambda

dimension a(m1),ja(m1),ia(m2),b(m3),jb(m3),ib(m2),c(m3),jc(m3),
1 ic(m2),l(m3),j1(m3),il(m2),u(m3),ju(m3),iu(m2),diag(m2),jdd(m2),
2 idd(m2),x(m2),z(m2),y(m2)

***********************************************************************
* * THIS SUBROUTINE EVALUATES THE EXTREME EIGENVALUES OF A * 
* REAL SYMMETRIC MATRIX A BY THE "SHIFT OF ORIGIN" METHOD DES- * 
* CRIBED IN (V.4.2). * 
* ***********************************************************************

INITIALISATION OF THE PARAMETERS
=====================================

ALL THE PARAMETERS USED IN THIS SUBROUTINE ARE DEFINED IN THE
PROGRAM "precond.f" EXCEPT

w - THIS PARAMETER REPRESENTS THE VALUE OF THE SHIFT WHICH
MUST BE GIVEN BY THE USER.

***********************************************************************
* * THE INTERNAL SUBROUTINES USED IN THIS PROGRAM ARE 
* * FACT.f SPADD.f SPSUB.f VECMUL.f FORSUB.f 
* * BAKSUB.f 
* * ***********************************************************************

n=m2-1
imark=0
eps=0.000001

10 format(f12.7)
20 format(i6)
do 40 i=1,n
   do 30 k=ia(i),ia(i+1)-1
      if(ja(k).ne.i) goto 30
      diag(i)=a(k)
goto 40
30 continue
40 continue
call fact(a,ja,ia,n,l,jl,il,u,ju,iu)
istepl=1
iter=0
istepl=istepl
eps=0.000001

MARK=0
WRITE(6,50) LAMBDA
      FORMAT(’THE INITIAL VALUE OF LAMBDA=’,F12.7)
WRITE(6,60)
      FORMAT(’********************************************************************’)
DO 150 I=1,5000
DO 70 K=1,N
DD(K)=DIAG(K)-LAMBDA
JDD(K)=K
IDD(K)=K
70 CONTINUE
IDD(N+1)=N+1
DO 80 K=1,150
C(K)=0.
B(K)=0.
JB(K)=0.
JC(K)=0.
80 CONTINUE
DO 90 K=1,N+1
15(K)=0.
IC(K)=0.
90 CONTINUE
CALL SSPSUB(DD,JDD,IDD,N,N,J1,I1,N,N,B,JB,IB)
B=(D-LAMBDA)-W*L.
CALL SSPSUB(DD,JDD,IDD,N,N,U,JU,IU,N,N,C,JC,IC)
C=(D-LAMBDA)-W*U.
IMARK=1
DO 260 J=1,ISTEP2
CALL VECMUL(U1,JU1,IU1,N,X,N,Z)
Z=U*X.
DO 100 K=1,N
Z(K)=Z(K)/DD(K)
100 CONTINUE
Z=INVERSE(D-LAMBDA)*Z.
CALL VECMUL(L1,DL1,IL1,N,Z,Y)
Y=L*Z     OR  Y=L*INVERSE(D-LAMBDA)*U*X.
DO 110 K=1,N
Z(K)=Y(K)
110 CONTINUE
CALL FORSUB(B,JB,BN1,Z)
Z=BS.
do 120 k=1,n
z(k)=z(k)*dd(k)
120 continue

call baksu(c;JC;JC;n,z)

Z=(D-LAMBDA)*Z.

130 continue

Z=C*Z

140 format(5(e25.15))
do 140 k=1,n
z(k)=z(k)-w*x(k)
140 continue

SHIFT

140 continue
xmax=0.0
do 150 k=1,n
if(abs(z(k)).lt.abs(xmax)) goto 150
xmax=z(k)
150 continue
if(xmax.eq.0.0) goto 180
160 format('XMAX=',f12.7)
do 170 k=1,n
z(k)=z(k)/abs(xmax)
170 continue
if(abs((x(k)-z(k))/(1.+abs(x(k)))).lt.eps) goto 190
goto 240
190 continue
iter=iter+j
write(6,320) lambda
write(6,210)
200 format('************** THE EIGENVECTOR **************','/)
write(6,230) (z(k);k=1,n)
210 format('ITER=',i6)
write(6,220) iter
write(6,230) lambda
220 format(5(F12.7,3X))
goto 350
230 format(5(F12.7,3X))
240 do 250 k=1,n
x(k)=z(k)
250 continue

260 continue
iter=iter+istep2
s1=0.0
do 270 k=1,n
s1=s1+x(k)*x(k)
270 continue

call vecmul(a;jas;ain;ni;x;in;z)

THE NEW EVALUATION OF THE RAYLEIGH QUOTIENT.
```
s=0.0
    do 280 k=1,n
        s=s+x(k)*z(k)
        continue
    if (mark.eq.1) goto 310
    op=s/s1
    if (abs(pp-lambda).gt.eps) goto 310
    write(6,290)
    290 format: 'THE E/VALUE HAS CONVERGED TO THE REAL ANSWER',/
        'AND THE E/VECTOR CORRESPONDING TO THIS VALUE IS GIVEN'
        'HOWEVER, THE PROCESS IS CONTINUED TO GET A BETTER ',/
        'ESTIMATE OF THE E/VECTOR IF iflag=1.' /
        write(6,300) iter
    300 format('THE NO. OF ITERATIONS SOFAR=',i6)
    write(6,320) (x(k)+k=1,n)
    mark=1
    eps=eps/1.
    step=5000
    if (iflag.eq.0) return
    lambda=s/s1
    write(6,130) lambda
    320 format('LAMBDA=',f12.7,/
    continue
    write(6,340)
    340 format('AFTER 5000 ITERATIONS DID NOT CONVERGED.')
    350 continue
    return
end```

subroutine sor(m1,m2,m3,iflag,a,ja,ia,l,jl,il,u,ju,iu,e,je,
  ie,f,jf,if,di,jd,id,dd1,dd2,z,x,lambda)

dimension a(m1),ja(m1),ia(m2),l(m3),jl(m3),il(m3),u(m3),ju(m3),
  iu(m2),e(m3),je(m3),ie(m2),f(m3),jf(m3),if(m2),d(m2),jd(m2),
  id(m3),dd1(m2),dd2(m2),z(m2),x(m2)

real l,lambda

***********************************************************************
* * THIS SUBROUTINE EVALUATES THE EXTREME EIGENVALUES * *
* AND THEIR CORRESPONDING EIGENVECTORS OF A REAL SYMMETRIC * *
* MATRIX A BY THE S.O.R METHOD DESCRIBED IN (V.3). * *
* ***********************************************************************

INITIALISATION OF THE PARAMETERS
==================================

THE PARAMETERS m1,m2,m3,iflags,x AND lambda ARE DEFINED IN THE
SUBROUTINE "precond.f" AND THESE MUST BE INITIALISED BY THE
USER.

THE MATRICES (a,ja,ia),(l,jl,il) AND (u,ju,iu) ARE ALSO DEFI-
NED IN THE SUBROUTINE "precond.f". THE MATRIX (a,ja,ia) REPRE-
SENTS THE COEFFICIENT MATRIX A AND MUST BE INITIALISED BY
THE USER. THE OTHER TWO MATRICES (l,jl,il) AND (u,ju,iu) ARE
GENERATED BY THE SUBROUTINE.

(e,je,ie) - THIS MATRIX IS USED FOR THE STORAGE OF THE SUCCESS-
ATIVE MATRICES (di-w1) AND IS EVALUATED BY THE
SUBROUTINE.

(f,jf,if) - THE SUCCESSIVE MATRICES (1-w)*di+wu ARE STORED IN
THIS MATRIX AND IS ALSO GENERATED BY THE SUBROUTINE.

w - IS THE ACCELERATION PARAMETER PROVIDED BY THE USER.

OTHER ARRAYS - ALL THE OTHER ARRAYS USED IN THIS SUBROUTINE
ARE WORKING SPACE. THE SIZE OF THESE ARRAYS
ARE GIVEN IN THE DIMENSION STATEMENT.

***********************************************************************
* THE INTERNAL SUBROUTINES USED IN THIS PROGRAM ARE: *
* FACT.F DMINADDWA.F FORSUB.F VECMUL.F *
*
n=m2-1
write(6,10) n
10 format('***** THE SIZE OF THE MATRIX =',i6,'*****',/)
mark=0
eps=0.000001
call fact(a,ja,ia,n,1,jl,il,u,ju,iu)
do 30 i=1,n
do 20 k=ia(i),ia(i+1)-1
if(ja(k).ne.i) goto 20
    d(i)=a(k)
goto 30
20 continue
30 continue
do 40 i=1,n
    jd(i)=i
    id(i)=i
40 continue
id(n+1)=n+1
mark=0
it=0
istep=1
write(6,50) w
50 format('THE VALUE OF w=',f12.7,/)  
write(6,60) lambda
60 format('THE INITIAL VALUE OF LAMBDA= ',f12.7,/)  
write(6,70)
70 format('**********************************************  
    ********7,1
write(6,70) lambda
70 format('**********************************************  
    ********7,1
do 300 iter=1,5000
do 80 i=1,n
    dd1(i)=d(i)-lambda
80 continue
call dminaddwa(l,jl,il,n,dd1,w,e,je,ie,1)
E=(D-LAMBDA)-w*L

    do 90 i=1,n
        dd2(i)=(1.-w)*(d(i)-lambda)
90 continue
call dminaddwa(u,ju,iiu,n,dd2,w,f,jf,if,2)
F=(1-w)*(D-LAMBDA)*wU

    do 210 j=1,istep
        write(6,18)
call vecmul(f,jf,if,n,n,x,n,z)
write(6,110) (z(i),i=1,n)

Z=((1-w)*(D-LAMBDA)+wU)*X

call forsub(e,je,ie,n,z)
Z IS THE NEW SOLUTION.
c

xmax=0.0
do 100 i=1,n
if(abs(z(i)).le.abs(xmax)) goto 100
xmax=z(i)
100 continue
if(xmax.eq.0.0) goto 120
do 110 i=1,n
z(i)=z(i)/abs(xmax)
110 continue
do 120 i=1,n
if(abs(x(i)-z(i))/((1.+abs(x(i))).lt.eps) goto 130
goto 190
120 continue
it=it+j
write(6,140)
140 format('**********THE E/VECTOR**********')
write(6,150) (z(i),i=1,n)
150 format(5(f12.7,i3x))
write(6,160)
160 format('THE E/VALUE CORRESPONDING TO THE ABOVE E/VECTOR')
write(6,170) lambda
170 format('THE E/VALUE=',f12.7)
write(6,180) it
180 format('THE NO. OF ITERATIONS=',i6)
go to 320
190 do 200 i=1,n
200 x(i)=z(i)
210 continue
it=it+istep
call vecmul(a,ja,ia,n,n,z,n,x)

THE NEW EVALUATION OF THE RAYLEIGH QUOTIENT.

x=A*z
s1=0.0
do 220 i=1,n
s1=s1+z(i)*x(i)
220 continue
s2=0.0
do 230 i=1,n
s2=s2+z(i)*z(i)
230 continue
pp=lambda
lambda=s1/s2
if(mark.eq.1) goto 280
if(abs(pp-lambda).gt.eps) goto 280
mark=1
write(6,240)
240 format('THE E/VALUE HAS CONVERGED TO THE REAL SOLUTION BUT THE'
1 'IF ifstream=1',/')
write(6,250) it
format('THE NO. OF ITERATIONS SOFAR ',i6)
write(6,260) lambda
format(/,'THE E/VALUE = ',f12.7)
write(6,270)
format('THE E/VECTOR CORRESPONDING TO THE ABOVE E/VALUE',/)
write(6,150) (z(i)+i=1,n)
istep=5000
if(iflag.eq.0) return
280 do 290 i=1,n
x(i)=z(i)
290 continue
300 continue
write(6,310)
format('AFTER 5000 ITERATIONS THE PROBLEM DID NOT CONVERGED')
320 continue
return
end
subroutine spadd(a, ja, ia, ma, na, b, jb, ib, mb, nb, c, jc, ic)
dimension a(1), ja(1), ia(1), b(1), jb(1), ib(1), c(1), jc(1), ic(1)

THIS SUBROUTINE PERFORMS THE MATRIX OPERATION A = B + C.

ma is the no. of rows in a.
na is the no. of cols. in a
mb is the no. of rows in b
nb is the no. of cols. in b

**********************************************************
**********************************************************
if(ma.ne.mb) goto 99
if(na.ne.nb) goto 999
noma1 = 1
nomb1 = 1
ic(1) = 1
nk = 1
kk = 1
do 1 i = 1, ma
noma2 = ia(i+1)
nomb2 = ib(i+1)
nna = noma2 - noma1
nnb = nomb2 - nomb1
if(nna.le.0.and.nnb.le.0) goto 10
if(nna.gt.0.and.nnb.le.0) goto 11
if(nna.le.0.and.nnb.gt.0) goto 12
j2 = noma1
j1 = nomb1
100 if(ja(j1).gt.jb(j2)) goto 13
   if(ja(j1).lt.jb(j2)) goto 14
   c(kk) = a(j1) + b(j2)
   if(c(kk).eq.0.0) goto 400
   jc(kk) = ja(j1)
   j1 = j1 + 1
   j2 = j2 + 1
   goto 1000

400 j1 = j1 + 1
     j2 = j2 + 1
     goto 410

14 c(kk) = a(j1)
    jc(kk) = ja(j1)
    j1 = j1 + 1
    goto 1000

13 c(kk) = b(j2)
    jc(kk) = jb(j2)
    j2 = j2 + 1

1000 kk = kk + 1
     nk = nk + 1

410 if(j1.lt.noma2.and.j2.lt.nomb2) goto 100
if(j1.lt.noma2.and.j2.ge.nomb2) goto 101
if(j1.ge.noma2.and.j2.lt.nomb2) goto 102
     goto 2000

102 do 30 iii = j2, nomb2 - 1
c(kk)=b(iii)
jc(kk)=jb(iii)
kk=kk+1
nk=nk+1
30 continue
goto 2000
101 do 20 iii=j1,noma2-1
c(kk)=a(iii)
jc(kk)=ja(iii)
kk=kk+1
nk=nk+1
20 continue
goto 2000
ic(i+1)=nk
noma1=noma2
nomb1=nomb2
2000 continue
ic(i+1)=nk
noma1=noma2
nomb1=nomb2
j2=nomb1
do 40 if=j2,nomb2-1
c(kk)=b(if)
jc(kk)=jb(if)
kk=kk+1
nk=nk+1
40 continue
ic(i+1)=nk
goto 350
j1=noma1
do 41 if=j1,noma2-1
c(kk)=a(if)
jc(kk)=ja(if)
kk=kk+1
nk=nk+1
41 continue
ic(i+1)=nk
goto 350
10 ic(i+1)=ic(i)
350 noma1=noma2
nomb1=nomb2
1 continue
goto 90
99 write(6,31)
31 format('the row numbers do not match.error no. 102')
goto 80
999 write(6,32)
32 format('the col. numbers do not match.error no. 101')
80 stop
90 return
end
spsub.f

subroutine spsub(a, ia, ma, na, b, jb, mb, nb, c, jc, ic)
dimension a(1), ia(1), ma(1), b(1), jb(1), mb(1), nb(1), c(1), jc(1), ic(1)

THIS SUBROUTINE PERFORMS THE MATRIX OPERATION A=B-C.
THE SUBROUTINE SPADD.F IS AN INTERNAL ROUTINE TO THIS
PROGRAM.

ma is no. of rows in a.
na is no. of cols. in a.
mb is no. of rows in b
nb is no. of cols. in b

*******************************************************************************
*******************************************************************************
do 1 i=1, ib(mb+1)-1
   b(i)=-b(i)
   continue
   call spadd(a, ia, ma, na, b, jb, mb, nb, c, jc, ic)
do 2 j=1, ib(mb+1)-1
   b(j)=-b(j)
   continue
return
end
subroutine vecmul(a, ja, ia, ma, na, v, mv, u)

THIS SUBROUTINE PERFORMS THE MATRIX OPERATION A*V WHERE A IS A MATRIX AND V IS A VECTOR.

dimension a(1), ja(1), ia(1), v(1), u(1)
ma is no. of rows in A.
na is no. of cols. in A.
mv is no. of elements in vector V.
vector U has been used for the result.

******************************************************************************
if(na.ne.mv) goto 99
do 1 i=1,ma
m=ia(i+1)-ia(i)
if(m.eq.0) goto 10
s=0.
do 2 j=ia(i),ia(i+1)-1
m=ja(j)
s=s+a(j)*v(m)
2 continue
u(i)=s
goto 1
10 u(i)=0.
1 continue
goto 100
99 write(6,5)
5 format('the no. of cols. in matrix # no. of elements in
1 the vector . Error no. 107')
stop
100 return
end
subroutine baksub(a, ja, ia, n, x)

THIS MATRIX PERFORMS THE PROCESS OF BACK SUBSTITUTION.

X IS CHANGED i.e... IN AX=B, X IS STORED ON B.

dimension a(1), ja(1), ia(1), x(1)
if(a(ia(n)).eq.0.0) goto 99
x(n)=x(n)/a(ia(n))
k=n-1
10
s=x(k)
if((ia(k+1)-ia(k)).eq.1) goto 20
do 1 j=ia(k)+1, ia(k+1)-1
s=s-a(j)*x(ja(j))
continue
1
if(a(ia(k)).eq.0.0) goto 99
20
x(k)=s/a(ia(k))
k=k-1
if(k.ge.1) goto 10
99
write(6,98)
98 format('ONE OF THE DIAGONAL ELEMENTS IS ZERO. Error no. 112')
97
return
end
subroutine forsub(a, ja, ia, n, x)

THIS SUBROUTINE PERFORMS THE PROCESS OF FORWARD SUBSTITUTION.

NOTE THE COMMENT ON BACKSUBS SUBROUTINE

dimension a(1), ja(1), ia(1), x(1)
if(a(ia(1)).eq.0.0) goto 99
x(1)=x(1)/a(ia(1))
do 1 k=2, n
s=x(k)
if((ia(k+1)-ia(k)).eq.1) goto 20
do 2 j=ia(k), ia(k+1)-2
s=s-a(j)*x(ja(j))
continue
2
if(a(ia(k+1)-1).eq.0.0) goto 99
x(k)=s/a(ia(k+1)-1)
1
continue
 goto 97
99 write(6,98)
98 format('ONE OF THE DIAGONAL ELEMENTS IS ZERO. Error no.112')
stop
97 return
end
subroutine fact(a, ja, ia, n, l, j1, il, u, ju, iu)
dimension a(1), ja(1), ia(1), l(1), j1(1), il(1), u(1), ju(1), iu(1)
real 1

THIS PROGRAM FACTORISES A GIVEN MATRIX A IN THE FORM;

A = D - L - U

WHERE L, U ARE THE LOWER AND UPPER TRINGULAR COMPONENTS OF
A AND D CONTAINS THE DIAGONAL ELEMENTS OF A.

il(1)=1
iu(1)=1
kk=1
nk=1
do 4 i=1, n
do 7 j=ia(i), ia(i+1)-1
if(ja(j).eq.i) goto 7
if(ja(j).gt.i) goto 11
l(nk)=-a(j)
jl(nk)=ja(j)
nk=nk+1
goto 7
11 u(kk)=-a(j)
ju(kk)=ja(j)
kk=kk+1
7 continue
il(i+1)=nk
iu(i+1)=kk
4 continue
return
end
subroutine dminaddwa(a, ja, ia, n, d, w, jb, ib, imark)
dimension a(1), ja(1), ia(1), d(1), b(1), jb(1) , ib(1)

THIS SUBROUTINE CALCULATES THE FORMULA $D + \text{or}-WA$ WHERE W IS PASSED AS A PARAMETER AND THE RESULT IS STORED IN (b, jb, ib).

IF $D + WA$ IS REQUIRED THEN IMARK=2 SHOULD BE INITIALIZED AND IF $D - WA$ IS REQUIRED THEN IMARK=1 MUST BE CHOSEN.

        ib(1)=1
        nk=1
        do 1 i=1,n
          if((ia(i+1)-ia(i)).ne.0) goto 30
          b(nk)=d(i)
          jb(nk)=i
          nk=nk+1
          goto 40
  30     k=0
        do 2 J=ia(i),ia(i+1)-1
          if(ja(J).gt.d) goto 10
          if(ja(J).eq.i) goto 20
          b(nk)=(-1)**imark*(w*a(J))
          if(b(nk).eq.0.0) goto 2
          jb(nk)=Ja(J)
          nk=nk+1
          goto 2
  20     b(nk)=d(i)+(-1)**imark*(w*a(J))
        k=1
        if(b(nk).eq.0.0) goto 2
        jb(nk)=ja(J)
        nk=nk+1
        goto 2
  10     if(k.ne.1) goto 100
        b(nk)=(-1)**imark*(w*a(J))
        jb(nk)=ja(J)
        nk=nk+1
        goto 2
 100    b(nk)=d(i)
        k=1
        jb(nk)=i
        nk=nk+1
        b(nk)=(-1)**imark*(w*a(J))
        if(b(nk).eq.0.0) goto 2
        jb(nk)=ja(J)
        nk=nk+1
        continue
        if(k.eq.1) goto 40
        b(nk)=d(i)
        jb(nk)=i
        nk=nk+1
  40     ib(i+1)=nk
        continue
        return
      end
subroutine vector(a, ja, ia, n, b, jb, ib, c, jc, ic, dd, x, z, iter)

dimension a(1), ja(1), ia(1), b(1), jb(1), ib(1), c(1), jc(1),
1 ic(1), dd(1), x(1), z(1)

THIS IS AN INTERNAL SUBROUTINE TO THE pcmin.f.

do 40 j=1,5000
   call vecmul(a, ja, ia, n, x, z)
call forsub(b, jb, ib, n)
do 520 k=1, n
   z(k)=z(k)*dd(k)
   continue
   call baksub(c, jc, ic, n, z)
do 130 k=1, n
   z(k)=x(k)-z(k)
   continue
   xmax=0.0
   do 140 k=1, n
      if(abs(z(k)).lt.abs(xmax)) goto 140
      xmax=z(k)
   continue
   do 150 k=1, n
      z(k)=z(k)/abs(xmax)
   continue
   do 120 k=1, n
      if(abs(z(k)-x(k))/(1.0+abs(x(k))).gt.0.00001) goto 400
   continue
   iter=iter+j
   return
400   do 410 k=1, n
      x(k)=z(k)
   continue
410   continue
40   continue
end