New preconditioning iterative methods for the solution of elliptic difference equations

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

Additional Information:

- A Master's Thesis. Submitted in partial fulfilment of the requirements for the award of Master of Philosophy at Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/28274

Publisher: © Carlos M. Camacho

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/

Please cite the published version.
This item was submitted to Loughborough University as an MPhil 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/
NEW PRECONDITIONING ITERATIVE METHODS FOR THE
SOLUTION OF ELLIPTIC DIFFERENCE EQUATIONS

BY

CARLOS M. CAMACHO

A thesis submitted to Loughborough University of Technology
in partial fulfilment of the requirements for the
award of Master of Philosophy (M.Phil.)

September, 1982.

SUPERVISORS: Professor D.J. Evans, D.Sc.,
A. Benson, Ph.D.

© Carlos M. Camacho, 1982.
NEW PRECONDITIONING ITERATIVE METHODS FOR THE
SOLUTION OF ELLIPTIC DIFFERENCE EQUATIONS

BY

CARLOS M. CAMACHO

A thesis submitted to Loughborough University of Technology
in partial fulfilment of the requirements for the
award of Master of Philosophy (M.Phil.)
September, 1982.

SUPERVISORS: Professor D.J. Evans, D.Sc.,
A. Benson, Ph.D.

© Carlos M. Camacho, 1982.
To B.J.R. and N.C.T.
ACKNOWLEDGEMENTS

The author would like to express his gratitude to the staff of the Computer Studies Department, Loughborough University, for their help and encouragement over the past year.

He is especially grateful to his supervisors, Professor D.J. Evans and Dr. A. Benson without whose guidance and patience the author would certainly have not been able to complete this thesis.

Last, but not least he would like to thank his parents to whom he is deeply indebted to, not only for their financial help, but also for their confidence.
ABSTRACT

In this thesis some new preconditioning strategies for obtaining the solution of large linear systems are introduced. An initial introduction deals with mathematical concepts and basic theory of direct and iterative methods which the reader may find helpful in the general appreciation of the thesis.

The theory of the preconditioned simultaneous method for symmetric matrices is then developed and extended to a Toeplitz matrix. Upper and lower bounds for the eigenvalue spectrum of the iteration matrix are derived and the analysis for the optimum preconditioning parameter and minimum condition number obtained. Experimental results are given which substantiate the theory presented.

The thesis further develops a new concept in preconditioning in which the Gauss-Jordan method is adapted for use on elliptic difference equations in a sparse iterative form resulting in the sparse product form of the inverse or SPPI being used as a conditioning matrix in an iteration formula. The theory is developed for various sparse matrices which occur in the numerical solution of boundary value problems.

Further, this explicit preconditioning strategy is incorporated into the well known and established first and second order stationary/non-stationary iterative methods so that a class of semi-iterative methods are formed. Again sufficient experimental results are included to show the advantages of using this approach in solving such problems.

Finally, some conclusions and suggestions for further work are given in the last chapter.
## CONTENTS

<table>
<thead>
<tr>
<th>Chapter 1: NUMERICAL SOLUTION OF ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS BY FINITE DIFFERENCE METHODS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Finite Difference Approximations to Derivatives</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Finite Difference Methods for Solving Elliptic P.D.E.'s</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Elliptic Problems with Irregular Points</td>
<td>8</td>
</tr>
<tr>
<td>1.4 Boundary Conditions</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 2: DIRECT METHODS FOR SOLVING LINEAR SIMULTANEOUS EQUATIONS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.0 Introduction</td>
<td>11</td>
</tr>
<tr>
<td>2.1 Gaussian Elimination</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Gauss-Jordan Elimination</td>
<td>12</td>
</tr>
<tr>
<td>2.3 Product Form of the Inverse</td>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 3: BASIC ITERATIVE METHODS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.0 Introduction</td>
<td>15</td>
</tr>
<tr>
<td>3.1 Stationary Linear Iterative Methods and Their Rates of Convergence</td>
<td>15</td>
</tr>
<tr>
<td>3.2 The Acceleration of Simultaneous Iteration Methods</td>
<td>18</td>
</tr>
<tr>
<td>3.3 Non-Stationary Iterative Methods</td>
<td>21</td>
</tr>
<tr>
<td>3.4 Second Order Iterative Methods</td>
<td>22</td>
</tr>
<tr>
<td>3.5 Successive Iterative Methods</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 4: THE USE OF PRECONDITIONING IN SOLVING A TOEPLITZ SYSTEM OF SIMULTANEOUS EQUATIONS</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.0 Introduction</td>
<td>26</td>
</tr>
<tr>
<td>4.1 The Preconditioning of a Toeplitz Matrix</td>
<td>27</td>
</tr>
<tr>
<td>4.2 The Preconditioned Simultaneous Displacement Iterative Method</td>
<td>31</td>
</tr>
<tr>
<td>4.3 Experimental Analysis</td>
<td>34</td>
</tr>
</tbody>
</table>
Chapter 5: THE SPARSE PRODUCT FORM OF THE INVERSE AND ITS USE IN ITERATIVE METHODS

5.0 Introduction 42
5.1 The Sparse Product Form of the Inverse 42
5.2 An Algorithm for Forming the SPFI 47
5.3 Choosing a Suitable Sparsity Factor 49
5.4 The SPFI Semi-Iterative Method 53
5.5 Experimental Analysis 54
5.6 Computational Comparison 57

Chapter 6: CONCLUSIONS

6.0 Introduction 59
6.1 The Use of Preconditioning in Solving a Toeplitz System of Simultaneous Linear Equations 59
6.2 The Sparse Product Form of the Inverse 60
6.3 Suggestions for Further Work 61

REFERENCES 63

Appendix 1: FORTRAN PROGRAMS ILLUSTRATING THE PRECONDITIONING OF A TOEPLITZ MATRIX 66

Appendix 2: FORTRAN PROGRAMS ILLUSTRATING THE USE OF THE SPFI APPROACH IN ITERATIVE METHODS 71
CHAPTER 1
NUMERICAL SOLUTION OF ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS BY FINITE DIFFERENCE METHODS
1.0 INTRODUCTION

Mathematical formulation of most physical situations involving rates of change with respect to more than one independent variable lead either to a partial differential equation (p.d.e.), or a set of such equations. The method and speed of solution of these partial differential equations are therefore of great importance.

Let \( R \) be a plane region which is bounded and connected, having a boundary \( C \) and let \( s(x,y) \) be a function defined on \( C \). The problem is now that of finding a function \( U(x,y) \) which is continuous in \( R+C \), twice differentiable and satisfying in \( R \), the general second order partial differential equation,

\[
\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial x \partial y} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial U}{\partial x} + \frac{\partial U}{\partial y} + fU + g = 0 ,
\]

and on \( C \) the condition,

\[
U(x,y) = s(x,y) , \text{ for } (x,y \in C)
\]

The equation (1.1) is linear if \( a,b,c,d,e,f,g \) are constants or functions of \( x \) and \( y \) only and is said to be:

ELLiptic if \( b^2 < 4ac \)
PARAbolic if \( b^2 = 4ac \)
HYPERBolic if \( b^2 > 4ac \)

Typical examples of these three different types of p.d.e. are:-

Poisson's equation \( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y) \) - ELLIPtic
Laplace's equation \( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \) - HYPERBolic

the wave equation, \( \frac{\partial^2 u}{\partial x^2} = \frac{1}{c^2} \frac{\partial^2 u}{\partial t^2} \) - HYPERBolic

the heat conduction equation \( \frac{\partial^2 u}{\partial x^2} = \frac{1}{K} \frac{\partial u}{\partial t} \) - PARABolic

As the solution of only elliptic p.d.e.'s has been considered in this research work, no further mention of parabolic or hyperbolic p.d.e.'s will be made in this report.
1.1 FINITE DIFFERENCE APPROXIMATIONS TO DERIVATIVES

Finite difference approximation methods are frequently used in the solution of elliptic partial differential equations because they are universally applicable to linear and non-linear problems. Derivatives at a point are approximated by difference quotients over a small interval, i.e. \( \frac{\partial u}{\partial x} \) is replaced by \( \frac{\delta u}{\delta x} \) where \( \delta x \) is small. In other words, the domain \( \mathbb{R} \) of independent variables is replaced by a finite set of points, called 'mesh points' and approximate values of the desired solution are sought at these points.

These finite difference approximations can be derived by the use of the Taylor Series expansion. This method is widely used and is probably the best known of all for deriving finite difference approximations.

An example of mesh points is shown in Figure 1.0 and in this case they are the points of intersection of lines drawn parallel to the co-ordinate axes. The distance(s) between the parallel lines will be called the mesh size(s).

Assume a constant mesh size \( h \) and also that \( U(x,y) \) is sufficiently differentiable in a large neighbourhood about a point \( (x,y) \). Then by
1.1 FINITE DIFFERENCE APPROXIMATIONS TO DERIVATIVES

Finite difference approximation methods are frequently used in the
solution of elliptic partial differential equations because they are
universally applicable to linear and non-linear problems. Derivatives at
a point are approximated by difference quotients over a small interval,
i.e. $\frac{\partial U}{\partial x}$ is replaced by $\delta u/\delta x$ where $\delta x$ is small. In other words, the
domain $R$ of independent variables is replaced by a finite set of points,
called 'mesh points' and approximate values of the desired solution are
sought at these points.

These finite difference approximations can be derived by the use of
the Taylor Series expansion. This method is widely used and is probably
the best known of all for deriving finite difference approximations.

An example of mesh points is shown in Figure 1.0 and in this case
they are the points of intersection of lines drawn parallel to the co-
ordinate axes. The distance(s) between the parallel lines will be called
the mesh size(s).

![Figure 1.0](image)

Assume a constant mesh size $h$ and also that $U(x,y)$ is sufficiently
differentiable in a large neighbourhood about a point $(x,y)$. Then by
Taylor's theorem we have:

\[ U(x+h,y) = U(x,y) + h \frac{\partial U}{\partial x} + \frac{h^2}{2!} \frac{\partial^2 U}{\partial x^2} + \frac{h^3}{3!} \frac{\partial^3 U}{\partial x^3} + \ldots \]  

(1.2)

\[ U(x-h,y) = U(x,y) - h \frac{\partial U}{\partial x} + \frac{h^2}{2!} \frac{\partial^2 U}{\partial x^2} - \frac{h^3}{3!} \frac{\partial^3 U}{\partial x^3} + \ldots \]  

(1.3)

The following approximations for derivatives at the point \((x,y)\) are derived:

from (1.2), the forward difference approximation,

\[ \frac{\partial U}{\partial x} = \frac{U(x+h,y)-U(x,y)}{h} + \mathcal{O}(h), \]  

(1.4)

from (1.3), the backward difference approximation,

\[ \frac{\partial U}{\partial x} = \frac{U(x,y)-U(x-h,y)}{h} + \mathcal{O}(h), \]  

(1.5)

and the central difference approximations,

\[ (1.2)-(1.3) \frac{\partial U}{\partial x} = \frac{U(x+h,y)-U(x-h,y)}{2h} + \mathcal{O}(h^2), \]  

(1.6)

\[ (1.2)+(1.3) \frac{\partial^2 U}{\partial x^2} = \frac{U(x-h,y)-2U(x,y)+U(x+h,y)}{h^2} + \mathcal{O}(h^2) \]  

(1.7)

Similarly,

\[ U(x,y+h) = U(x,y) + h \frac{\partial U}{\partial y} + \frac{h^2}{2!} \frac{\partial^2 U}{\partial y^2} + \frac{h^3}{3!} \frac{\partial^3 U}{\partial y^3} + \ldots \]  

(1.8)

which yields the approximations,

\[ \frac{\partial U}{\partial y} = \frac{U(x,y+h)-U(x,y-h)}{2h} + \mathcal{O}(h^2) \]  

(1.9)

\[ \frac{\partial^2 U}{\partial y^2} = \frac{U(x,y+h)-2U(x,y)+U(x,y-h)}{h^2} + \mathcal{O}(h^2) \]  

(1.10)

The formulae (1.2-1.10) can be re-written so that they can be applied to any mesh point.
If $x = ih$

and $y = jh$,

then (1.4) can be re-written as,

$$\frac{\partial U}{\partial x} = \left(\frac{\partial U}{\partial x}\right)_{i,j} = \frac{U_{i+1,j} - U_{i,j}}{h} + O(h)$$  \hspace{1cm} (1.11)

and (1.5-1.7) as,

$$\left(\frac{\partial U}{\partial x}\right)_{i,j} = \frac{U_{i+1,j} - U_{i-1,j}}{h} + O(h)$$  \hspace{1cm} (1.12)

$$\left(\frac{\partial^2 U}{\partial x^2}\right)_{i,j} = \frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} + O(h^2)$$  \hspace{1cm} (1.13)

Similarly for the $y$-derivatives,

$$\left(\frac{\partial U}{\partial y}\right)_{i,j} = \frac{U_{i,j+1} - U_{i,j-1}}{2h} + O(h)$$  \hspace{1cm} (1.15)

$$\left(\frac{\partial^2 U}{\partial y^2}\right)_{i,j} = \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h^2} + O(h^2)$$  \hspace{1cm} (1.16)

replace (1.9-1.10) as a finite difference approximation for a general mesh point.
1.2 FINITE DIFFERENCE METHODS FOR SOLVING ELLIPTIC P.D.E.'S

The finite difference approximations derived in Section 1.1 can now be used to solve partial differential equations. An example of how they are used in solving elliptic p.d.e's is seen by solving Poisson's equation,

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = f(x,y)$$  \hspace{1cm} (1.17)

in a unit square $R$ with $0 \leq x, y \leq 1$.

Mesh size $h = 1/4$

and $U = G(x,y)$ on the boundary.

The only unknown points are $1, 2, 3, \ldots, 9$.

![Figure 1.2](image)

Replace the derivatives in Poisson's equation by the finite difference approximations, (1.14) and (1.16) at each unknown point $(x_i, y_j)$. Therefore,

$$\frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h^2} + \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{h^2} = \frac{P_{i,j}}{h^2}$$  \hspace{1cm} (1.18)

where $P_{i,j} = f(x_i, y_j) + \frac{h^2}{12} \left( \frac{\partial^4 U}{\partial x^4} + \frac{\partial^4 U}{\partial y^4} \right)_{i,j} + \ldots$

The terms on the right hand side of the above equation, excluding $f(x_i, y_j)$ are known as the local truncation error, and the $O(h^2)$ term is defined as the principal part of this error. It can be shown how this term is obtained by looking more closely at equations (1.7) and (1.10).

In deriving (1.7) for example, the local truncation error was written down as $O(h^2)$. Writing the $O(h^2)$ term in (1.7) more fully gives,

$$-\frac{h^2}{12} \frac{\partial^4 U}{\partial x^4} + \frac{h^4}{120} \frac{\partial^6 U}{\partial x^6} + \ldots$$
Doing the same for the $O(h^2)$ term in (1.10) gives,

$$h^2 \left( \frac{3^4 U}{4} \frac{\partial^4 U}{\partial y^4} + \frac{h^4}{120} \frac{\partial^6 U}{\partial y^6} + \ldots \right)$$

Therefore, when replacing the derivatives in (1.17) by finite difference approximations, a local truncation error of the form

$$\frac{h^2}{12} \left( \frac{3^4 U}{4} \frac{\partial^4 U}{\partial x^4} + \frac{h^4}{120} \frac{\partial^6 U}{\partial x^6} + \frac{\partial^6 U}{\partial y^6} \right) + \ldots$$

is obtained. For greater accuracy $O(h^2)$ terms may be included when deriving finite difference formulae, but when $h$ is small even terms of order $h^2$ are negligible and so these terms are not usually included.

Hence rewriting (1.18) without the local truncation error gives a five point difference approximation to Poisson's equation over a unit square with known boundary conditions as,

$$4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = -h^2 f(x_i, y_j)$$

(1.19)

where $u_{i,j}$ is the exact solution of the difference equation at the point $(i,j)$.

Equation (1.19) can be represented by the stencil,

![Stencil](image)

FIGURE 1.3

which when applied to each unknown mesh point, yields a set of independent...
simultaneous, non-homogeneous, linear equations. These can be represented as follows:

At point 1: 

\[ 4u_1 - u_2 - u_4 - u_{10} - u_{24} = h^2 f_1 \]
\[ 2: -u_1 + 4u_2 - u_3 - u_5 - u_{23} = h^2 f_2 \]
\[ 3: -u_2 + 4u_3 - u_6 - u_{19} - u_{22} = h^2 f_3 \]
\[ 4: -u_1 + 4u_4 - u_5 - u_7 - u_{11} = h^2 f_4 \]
\[ \vdots \]
\[ 9: -u_6 - u_8 + 4u_9 - u_{16} - u_{18} = h^2 f_9 \]

These equations can be written in matrix form with the known factors on the right hand side.

\[
\begin{bmatrix}
4 & -1 & 1 & -1 \\
-1 & 4 & -1 & 1 \\
-1 & 4 & -1 & 1 \\
-1 & 4 & -1 & 1 \\
-1 & 4 & -1 & 1 \\
-1 & 4 & -1 & 1 \\
-1 & 4 & -1 & 1 \\
-1 & 4 & -1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_2 \\
u_3 \\
u_4 \\
u_5 \\
u_6 \\
u_7 \\
u_8 \\
\end{bmatrix}
= h^2
\begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4 \\
f_5 \\
f_6 \\
f_7 \\
f_8 \\
\end{bmatrix}
\]

\[
A \quad \ast \quad u \quad = \quad b
\]

This system can now be solved by either a direct method, e.g. Gauss elimination or by an iterative method e.g. simultaneous displacement, S.O.R. etc. These methods are dealt with in subsequent chapters.
1.3 **ELLIPTIC PROBLEMS WITH IRREGULAR POINTS**

Irregular points occur when a boundary curve intersects the sides of the square meshes as in Fig. 1.0. They are points at which one or more of the four neighbouring points are distance less than \( h \) away (where \( h \) is the mesh size).

![FIGURE 1.4](image)

Irregular points are harder to deal with than rectangular points but it is still possible to obtain finite difference approximations at these points. As an example of how these approximations are obtained, consider irregular point \( p \) in Fig. 1.4.

By Taylor's expansion,

\[
U_A = U(x+s_1 h, y)
\]

\[
= U_p + \frac{s_1 h (\frac{\partial U}{\partial x})}{2} + \frac{(s_1 h)^2}{2} \left( \frac{\partial^2 U}{\partial x^2} \right) + \ldots
\]

(1.20)

\[
U_3 = U_p - h \left( \frac{\partial U}{\partial x} \right) + \frac{h^2}{2} \left( \frac{\partial^2 U}{\partial x^2} \right) + \ldots
\]

(1.21)

(1.20)-(1.21)*\( s_1^2 \) yields,

\[
\left( \frac{\partial U}{\partial x} \right)_p = \frac{1}{h} \left\{ \frac{U_A}{s_1 (1+s_1)} - \frac{s_1 U_3}{1+s_1} - \frac{(1-s_1)U_0}{s_1} \right\} + O(h^2)
\]

(1.22)

Also (1.20)+\( s_1 \)(1.21) gives,

\[
\left( \frac{\partial^2 U}{\partial x^2} \right)_p = \frac{2}{h^2} \left\{ \frac{U_A}{s_1 (1+s_1)} + \frac{U_3}{1+s_1} - \frac{U_0}{s_1} \right\} + O(h)
\]

(1.23)
1.4 **BOUNDARY CONDITIONS**

So far in this report it has been assumed that $U$ is known on the boundary of a plane connected region $R$. However, there are other types of boundary conditions which may arise. Three types of boundary conditions which may occur are:

1. $U$ given — DIRICHLET
2. $\frac{\partial U}{\partial x}$ given — NEUMANN
3. $\alpha U + \frac{\partial U}{\partial x} = \gamma$ — mixed or ROBIN'S

The DIRICHLET condition has already been dealt with in Section 1.2.

To show how NEUMANN conditions are dealt with, consider the problem of solving Poisson's equation (1.17) on the unit square in Fig. 1.5 with Neumann conditions on the boundary $x=0$, $0 \leq y \leq 1$.

![FIGURE 1.5](image-url)

Mesh points 4-12 are dealt with as before. For example, using the five point difference approximation, (1.19) at point 5 would yield,

$$4u_5 - u_2 - u_6 - u_4 - u_8 = -h^2 f_5$$  \hspace{1cm} (1.24)

The problem occurs at the boundary points 1, 2, 3 and is overcome by setting up a fictitious point at each of the points on the boundary. Consider point 2 as an example. A fictitious point $P$ is set up as shown in Fig. 1.5. Hence, when the five point difference approximation (1.19)
is applied at point 2 the following equation is obtained,

\[ 4u_2 - u_5 - u_1 - u_3 = -h^2 f_2 \] (1.25)

From the boundary condition it is known that,

\[ \left( \frac{\partial u}{\partial x} \right)_2 = g_2. \]

Therefore, by using the approximation (1.13) at point 2 and neglecting \( O(h^2) \) terms, the approximation,

\[ g_2 = \left( \frac{\partial u}{\partial x} \right)_2 \approx \frac{u_5 - u_p}{2h} \] (1.26)

is obtained, and so,

\[ u_p = u_5 - 2hg_2 \] (1.27)

By substituting this approximation of \( u_p \) into (1.25) gives a four point finite difference approximation,

\[ 4u_2 - u_3 - 2u_5 - u_1 = -2hg_2 - h^2 f_2 \] (1.28)

at point 2.

Thus the four-point finite difference approximation formula at any point \((i,j)\) on the boundary as given in Fig. 1.5 can thus be written as,

\[ 4u_{i,j} - u_{i,j+1} - 2u_{i-1,j} + u_{i-1,j-1} = -2h(x_i,y_j) - h^2 f(x_i,y_j) \] (1.29)

(and so can be applied to points 1-3).

MIXED conditions are treated in almost the same way as Neumann conditions. For example, assuming that the unit square in Fig. 1.5 has mixed conditions,

\[ \alpha u + \beta \frac{\partial u}{\partial x} = \gamma \]

on the boundary, \( x=0, 0 \leq y \leq 1 \) then at point 2

\[ \alpha_2 u_2 + \beta_2 \left( \frac{u_5 - u_p}{2h} \right) = \gamma_2 \]

A value for \( u_p \) is then found and substituted into (1.25), in a similar manner as for Neumann conditions. A finite difference approximation formula is thus obtained and this can be applied at all the points on the boundary of the square.
CHAPTER 2

DIRECT METHODS FOR SOLVING LINEAR

SIMULTANEOUS EQUATIONS
2.0 INTRODUCTION

It was shown in Chapter 1 that the use of finite difference methods to solve elliptic partial differential equations leads to a system of linear, simultaneous equations. This system of equations may be solved by using either direct or iterative methods; the purpose of this chapter is to give a brief outline of two types of direct methods.

One of the most popular techniques for solving a set of simultaneous equations is known as Gaussian elimination. Because of its popularity, it would be futile to go into a full explanation in this report and so only a brief algorithm is given. However, another less popular direct method, known as the Gauss-Jordan method, is explained fully. This method is used in later chapters and so knowledge of it is helpful.

2.1 GAUSSIAN ELIMINATION

To allow generality consider the following simultaneous equations,

\[
\begin{align*}
    a_{11}u_1 + a_{12}u_2 + a_{13}u_3 &= b_1 \\
    a_{21}u_1 + a_{22}u_2 + a_{23}u_3 &= b_2 \\
    a_{31}u_1 + a_{32}u_2 + a_{33}u_3 &= b_3
\end{align*}
\]

(2.1)

The basic method of Gauss reduces the equations to the triangular form,

\[
\begin{align*}
    a_{11}u_1 + a_{12}u_2 + a_{13}u_3 &= b_1 \\
    a_{22}u_2 + a_{23}u_3 &= b_{2(1)} \\
    a_{33}u_3 &= b_{3(2)}
\end{align*}
\]

(2.2)

The third equation then gives \( u_3 = b_3^{(2)}/a_{33}^{(2)} \), assuming \( a_{33}^{(2)} \neq 0 \). Back substitution in the second equation now gives \( u_2 \), assuming \( a_{22}^{(1)} \neq 0 \). Finally, back substitution in the first equation gives \( u_1 \), assuming \( a_{11} \neq 0 \).

The following is an algorithm which will perform Gaussian elimination on a set of \( N \) linear, simultaneous equations.
For \( j = 1, 2, \ldots, N - 1 \), repeat steps 1-5

1. Find the smallest index \( p \) such that \( |a_{pj}| = \max |a_{ij}| \) for \( i > j \)
2. If \( a_{pj} = 0 \), skip steps 3-5 for this value of \( j \)
3. If \( p > j \), interchange elements \( j, \ldots, N \) of rows \( j \) and \( p \) and similarly row \( p \) with row \( j \) of the RHS vector \( b \).
4. For \( i = j + 1, \ldots, N \) repeat
   
   Compute the multipliers: \( a_{ij} \rightarrow a_{ij}/a_{jj} \).
5. For \( i = j + 1, \ldots, N \) repeat

   \( t + a_{ij} \)

   if \( t \neq 0 \) row \( i \) \(- \) row \( i \) \(- t \times \text{row } j \) (only elements \( j+1, \ldots, N \) in each row).

   RHS row \( i \) \(+ \) RHS row \( i \) \(- t \times \text{RHS row } j \).

Alternatively step 5 may be replaced by

5. \(^*\) \( t + a_{ij} \), \( i = j + 1, \ldots, N; \)

   if \( t \neq 0 \) then \( \text{col } i \) \(+ \) \( \text{col } i \) \(- t \times \text{col } j \) (only elements \( j+1, \ldots, N \) in each column).

The first version is known as elimination by rows, (the second, elimination by columns) and incorporates the concept of partial pivoting which eliminates the theoretical problem of a pivot's \( a_{jj} \) in step 4) becoming zero at any stage and also is important from a numerical viewpoint in that the multipliers (in step 4 always have absolute value not greater than unity.

In the above algorithm, it should be noted that when a new element is computed it is stored in the space vacated by the eliminated element. This is also true for the Gauss–Jordan algorithm shown in the next section.

### 2.2 GAUSS JORDAN ELIMINATION

The system of equations (2.1) can be represented by the matrix equation

\[
Au = b
\]

where,

\[
A = \begin{bmatrix}
a_{11} & a_{12} & a_{13} \\
a_{21} & a_{22} & a_{23} \\
a_{31} & a_{32} & a_{33}
\end{bmatrix}
\]

\[
u = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}
\]

\[
b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}
\]
The general idea of Gauss-Jordan elimination is to reduce the matrix of coefficients i.e. A, to an identity matrix, which allows the values of \( u_1, u_2, u_3 \) to be determined immediately.

If, instead of these there were \( N \) linear equations to be solved then A would be an \( (N \times N) \) matrix and the following algorithm could be used to perform Gauss-Jordan elimination on the matrix A and the right hand side vector \( \mathbf{b} \).

For \( i=1,2,\ldots,N \) repeat steps 1-7
1. For \( \ell=i,i+1,\ldots,N \) repeat steps 2-3
2. \[ a_{i,\ell}^{(i)} = \frac{a_{i,\ell}^{(i-1)}}{a_{i,i}^{(i-1)}} \]
3. \[ b_{i}^{(i)} = \frac{b_{i}^{(i-1)}}{a_{i,i}^{(i-1)}} \]
4. For \( j=1,2,\ldots,N, \; i\neq j \) repeat steps 5-7
5. \[ b_{j}^{(i)} = b_{j}^{(i-1)} - b_{i}^{(i)} \times a_{j,i}^{(i-1)} \]
6. For \( k=i+1,N \) repeat 7
7. \[ a_{i,k}^{(i)} = a_{i,k}^{(i-1)} - a_{i,i}^{(i-1)} \times a_{j,i}^{(i-1)} \]

N.B. It is assumed that \( a_{ii} \neq 0 \).

It should be noted that the same operations that are performed on matrix A are also performed on vector \( \mathbf{b} \) and so after the algorithm has been completed the solution of \( \mathbf{A} \mathbf{u} = \mathbf{b} \) can be found in vector \( \mathbf{b} \).

2.3 PRODUCT FORM OF THE INVERSE

Let \( C^{(i)} \) be a square matrix which differs from a unit matrix only in the \( i \)th column and is of the form:

\[
C^{(i)} = \begin{bmatrix}
1 & -a_{i,1}^{(i-1)}/a_{i,i}^{(i-1)} \\
& 1 \\
& & \ddots \\
& & & 1 & -a_{i,N}^{(i-1)}/a_{i,i}^{(i-1)} \\
& & & & 1/a_{i,i}^{(i-1)} \\
& & & & & \ddots \\
& & & & & & 1
\end{bmatrix}
\]
Steps 1-7 excluding 3 and 5 in the Gauss-Jordan algorithm are then equivalent to the matrix product $C(i)A(i)$ where $A(i)$ is the state of matrix $A$ at the $i$th stage of the elimination process and $A(0)=A$.

The Gauss-Jordan algorithm may then be expressed as a set of matrix products,

$$A^{(0)} = A$$
$$A^{(1)} = C(1)A^{(0)} = C(1)A$$

and in general,

$$A^{(k)} = C(k)A^{(k-1)}$$

The final result is

$$A^{(N)} = C(N)C(N-1)\ldots C(1)A = I$$

where $I$ is the $N \times N$ unit matrix.

Since each $C(i)$ differs from a unit matrix only in one column, it requires just $N$ storage cells, and the product form of the inverse (PFI) requires $N^2$ cells just as the true inverse. The PFI for matrix $A$ is thus of the form:

$$PFI(A) = \begin{bmatrix}
\frac{1}{a_{11}} & -\frac{a(1)}{a_{12}} & \ldots & -\frac{a(i-1)}{a_{ii}} & \ldots & -\frac{a(N-1)}{a_{NN}} \\
\frac{-a(0)}{a_{21}} & \frac{1}{a_{22}} & \ldots & \frac{1}{a_{ii}} \\
\frac{-a(0)}{a_{31}} & \frac{-a(1)}{a_{32}} & \ldots & \frac{-a(i)}{a_{ii}} \\
\frac{-a(0)}{a_{N1}} & \frac{-a(1)}{a_{N2}} & \ldots & \frac{-a(N-1)}{a_{NN}}
\end{bmatrix}$$

The solution of $Au=b$ using the PFI can then be found by the sequence of matrix products,

$$u = A^{-1}b = C(N)C(N-1)\ldots (1)b$$

This method is as efficient for solving linear equations as is the true inverse. In both cases the solution of $Au=b$ requires $N^2$ multiplications.

The PFI may be thought of as a by-product of the Gauss-Jordan elimination process.
The PFI is obtained as a direct consequence of the Gauss-Jordan method and so interchanges for both are performed in the same way. The pivots for the Gauss-Jordan method can be chosen in the same manner as in the method of Gauss (§2.1). However, in this case elements from previous pivotal rows are also eliminated. The pivots themselves are unaffected since the other elements in previous pivotal columns are zero. It should be noted that there is no simple pivotal strategy which ensures that all row multipliers have values less than one.

The matrices considered in later chapters are diagonally dominant and positive definite and so no pivoting strategy is necessary and hence no further mention will be made of pivoting for the Gauss-Jordan method and the Product Form of the Inverse.

In conclusion, some mention must be made of how the Gauss and the Gauss-Jordan methods compare with each other in efficiency. For a full \((N \times N)\) matrix it is well known that the computational work of these methods in solving a system of linear equations as described earlier, is respectively \(O(N^3/3)\) and \(O(N^3/2)\) so that the Gauss-Jordan is more expensive by a factor of \(3/2\). In finding the inverse of a matrix, however both methods require the same amount of work.

Later in this report, sparse banded systems are considered and for coefficient matrices of this type with a half-bandwidth \(r\), the number of multiplications in the elimination stage becomes \(O(Nr^2)\) [Walsh (1962)] for Gauss elimination but \(O(N^2r)\) for the Gauss-Jordan method.
CHAPTER 3

BASIC ITERATIVE METHODS
3.0 INTRODUCTION

Iteration is a systematic method of generating successive approximations to one or more unknowns, starting with assumed values for the unknowns.

Consider the following system which was derived in §1.2,

\[ Au = b \]  

A sequence \( u^{(n)} \) is defined with the hope that as \( n \to \infty \), \( u^{(n)} \to u \), the true result, i.e.,

\[ u^{(n)} \to A^{-1}b, \quad \text{as } n \to \infty . \]

If \( u^{(n)} \) is a function of \( A, b, u^{(n-1)}, u^{(n-2)}, \ldots, u^{(n-r)} \) we say r is the degree of the iteration. To minimise computer storage requirements r is usually chosen as 1 or 2.

Putting \( r = 1 \) gives

\[ u^{(n)} = f_n(A, b, u^{(n-1)}). \]

If \( f_n \) is independent of \( n \), the iteration is said to be stationary. If \( f_n \) is linear in \( u^{(n-1)} \) the iteration is termed linear.

3.1 STATIONARY LINEAR ITERATIVE METHODS AND THEIR RATES OF CONVERGENCE

Any stationary linear iteration generates a sequence of vectors

\[ v^{(n)} = \{ v_1^{(n)}, v_2^{(n)}, \ldots, v_N^{(n)} \} , \]

by means of a formula of the form,

\[ v^{(n)} = H v^{(n-1)} = H^n v^{(0)} , \quad n=1,2,\ldots \]  

where \( H \) is an \( N \times N \) matrix. The convergence of the process therefore depends on the behaviour of \( H^n \) as \( n \to \infty \).

The basic problem is to solve (3.1) by an iterative process.

Put

\[ A = E - F , \]

so that

\[ Eu = Fu + b . \]  

By introducing an iteration count into (3.3) an iteration process may be formulated. Thus,
\[ Eu^{(n+1)} = Fu^{(n)} + b, \quad (3.4) \]

where \( u^{(n)} \) is the \( n \)th approximation to the true value \( u \). \( u^{(0)} \) is set to some arbitrary column vector.

\( E \) is chosen so that \( u^{(n+1)} \) is easily obtained from \( u^{(n)} \) in (3.4). Hence it is essential that \( \det(E) \) is non-zero and the inverse of \( E \) exists.

\( E \) is usually chosen as either a diagonal matrix, which gives rise to methods of simultaneous displacement (e.g. Jacobi, Richardson, etc.) or a lower or upper triangular matrix, which leads to methods of successive displacements (e.g. Gauss-Seidel, Successive Over-relaxation etc.). This report deals mainly with Simultaneous methods and so, only these methods (i.e. when \( E \) is a diagonal matrix) will be considered in detail.

Define the correction vector \( c \) by

\[ c^{(n)} = u^{(n+1)} - u^{(n)}, \quad (3.5) \]

and assume \( u^{(n)} \to u \) as \( n \to \infty \).

\[ \therefore \quad u = u^{(0)} + \sum_{n=1}^{\infty} c^{(n)}. \quad (3.6) \]

A necessary and sufficient condition for the convergence of the infinite series in (3.6) is

\[ c^{(n)} \to 0 \quad \text{as} \quad n \to \infty \]

From (3.4)

\[ Eu^{(n)} = Fu^{(n-1)} + b \]
\[ Eu^{(n+1)} = Fu^{(n)} + b \]

Subtracting the above two equations gives

\[ Ec^{(n+1)} = Fc^{(n)}, \]

or

\[ c^{(n+1)} = Kc^{(n)} = K^{n+1} c^{(0)} \quad (3.7) \]

where \( K = E^{-1}F \).

(3.7) is very similar to (3.2) with \( H = K = E^{-1}F \).

If \( c^{(n)} \to 0 \) as \( n \to \infty \) then it is necessary that \( |\lambda_i| < 1 \), where \( \lambda_i \) are the eigenvalues of \( H = E^{-1}F \) and
\[ |\lambda_1| > |\lambda_2| > \ldots |\lambda_N| \]

Young (1954a) introduces the number
\[ R = -\log_e (|\lambda_1|) \]
\[ = -\log_{10} (|\lambda_1|) \log_e 10 \]
(3.8)
(3.9)
as the rate of convergence of linear iterative methods.

Hence the smaller the value of \( |\lambda_1| \), the greater the rate of convergence. Therefore, for iterative processes to be most efficient, \( |\lambda_1| \) must be reduced to a minimum
\[ |\lambda_1| \]
is known as the spectral radius of \( K \).

Going back now to equation (3.4) and taking \( E \) as the identity matrix \( I \) (a diagonal matrix) then
\[ A = E - F \]
now becomes
\[ A = I - L - U \]
where \( F = L + U \) and \( L, U \) are lower and upper triangular matrices, respectively, with zero diagonal elements. Hence, (3.4) can now be written as,
\[ u^{(n+1)} = (L+U)u^{(n)} + b \]  
(3.10)
The iterative scheme (3.10) is known as Jacobi iteration and falls into the class of methods known as simultaneous displacement methods, i.e., all the iterates of \( u \) are advanced simultaneously and are entirely dependent on the previous iterate.

The difference, residual and error vectors are defined, respectively, by
\[ \Delta u^{(n)} = u^{(n+1)} - u^{(n)} \]
\[ r^{(n)} = b - (I-L-U)u^{(n)} \]
\[ e^{(n)} = u^{(n)} - u \]  
(3.11)
The sequence \( \{u^{(n)}\} \) converges to \( u \) only if the above three equations converge to zero. The equations (3.11) satisfy
\[ \Delta u^{(n+1)} = (L+U)\Delta u^{(n)} \]
\[ r^{(n+1)} = (L+U)r^{(n)} \]
and so the Jacobi iteration (3.10) converges if, and only if, the eigenvalues of \((L+U)\) are all less than unity in modulus.

The following are some useful results on the convergence of iteration schemes. The proofs have been omitted but if required may be found in the given references.

(1) - Geiringer (1949)

For a non singular matrix \(A=I-L-U\) the method of simultaneous iteration converges for an arbitrary starting point and for any order of solution of the equations if

\[
\max_i \sum_j |a_{ij}| < 1 , \ i \neq j, \text{ with strict inequality for at least one } i
\]

Defn.

A matrix \(A=(a_{i,j})\) is said to be positive definite if

\[
\sum_{i=1}^{N} \sum_{j=1}^{N} a_{i,j} x_i x_j > 0 \text{ for all } x \neq 0
\]

(2) - Bodewig (1956)

Given a real symmetric, positive definite matrix \(A=I-L-U\) the simultaneous iteration converges if the norm of \((L+U)\) is less than or equal to unity e.g.

\[
\|L+U\| = \sqrt{\sum_{i=1}^{N} \sum_{j=1}^{N} |a_{i,j}|^2} \leq 1
\]

where the Euclidean norm has been used.

3.2 THE ACCELERATION OF SIMULTANEOUS ITERATION METHODS

The Jacobi method of iteration (3.10) can be written as

\[
u^{(n+1)} = u^{(n)} + r^{(n)}
\]  

(3.12)

where \(r\) is the residual vector defined by:

\[
r^{(n)} = b-Au^{(n)}
\]
It has been stated that the rate of convergence of an iterative method depends on the spectral radius of the iteration matrix being made as small as possible. However, for a given A, the spectral radius of \((L+U)\) in (3.10) is fixed and there is no chance of minimisation in any way; consequently some means is sought of introducing a variable parameter into the iteration matrix so that with a judicious choice, the spectral radius might be minimised.

By introducing either a constant acceleration parameter \(\alpha\) or a different acceleration parameter \(\alpha_n\) for each iteration into (3.12) the following equations are obtained respectively,

\[
\begin{align*}
\mathbf{u}(n+1) &= \mathbf{u}(n) + \alpha \mathbf{r}(n) \\
\mathbf{u}(n+1) &= \mathbf{u}(n) + \alpha \mathbf{r}(n)
\end{align*}
\]  
(3.13)

Equations (3.13) may be re-written as

\[
\begin{align*}
\mathbf{u}(n+1) &= (I-\alpha A)\mathbf{u}(n) + \alpha \mathbf{b} \\
\mathbf{u}(n+1) &= (I-\alpha A)\mathbf{u}(n) + \alpha \mathbf{b}
\end{align*}
\]  
(3.14)

and

\[
\begin{align*}
\mathbf{u}(n+1) &= (I-\alpha A)\mathbf{u}(n) + \alpha \mathbf{b} \\
\mathbf{u}(n+1) &= (I-\alpha A)\mathbf{u}(n) + \alpha \mathbf{b}
\end{align*}
\]  
(3.15)

(3.14) is known as the Accelerated Jacobi Method and (3.15), Richardson's Method.

The error vector associated with the scheme (3.14) can be shown to satisfy

\[
\begin{align*}
\mathbf{e}(n+1) &= (I-\alpha A)\mathbf{e}(n) \\
\mathbf{e}(n+1) &= (I-\alpha A)\mathbf{e}(n)
\end{align*}
\]  
(3.16)

or

\[
\begin{align*}
\mathbf{e}(n+1) &= (I-\alpha A)^{n+1}\mathbf{e}(0) \\
\mathbf{e}(n+1) &= (I-\alpha A)^{n+1}\mathbf{e}(0)
\end{align*}
\]  
(3.17)

The method is thus a stationary, linear, iterative process as the error operator is constant throughout the iteration.

Assume the matrix \((I-\alpha A)\) has \(N\) linearly independent eigenvectors \(v_i\), associated with \(N\) distinct eigenvalues \(\lambda_i\). For the iteration to converge the modulus of the largest eigenvalue of \((I-\alpha A)\) must be less than unity.

Assume that the eigenvalues of \(A\), say \(\mu_i\), are bounded by values \(a, b\) so that,

\[
0 < a \leq \mu_i \leq b < \infty \quad (i=1,2,\ldots,N) .
\]  
(3.18)
Then \( \lambda_i = 1 - \alpha u_i \), \((i=1,2,\ldots,N)\)

and so for convergence

\[ |1 - \alpha u_i| < 1, \quad (i=1,2,\ldots,N) \]  \hspace{1cm} (3.19)

i.e.

\[ -1 < (1 - \alpha u_i) < 1 \]

and so

\[ 0 < \alpha < \frac{2}{b} \]

gives the range of values for \( \alpha \).

![Figure 3.1](image)

The fastest rate of convergence is obtained by choosing \( \alpha \) so that the spectral radius of \((I-\alpha A)\) is minimised.

The best value of \( \alpha \) is therefore that for which

\[ \max\{|1-\alpha a|,|1-ab|\} \]

is minimised

and this occurs when,

\[ 1-\alpha a = -(1-ab) \] as shown in Figure 3.1

\[ \therefore \quad \alpha = \frac{2}{a+b} \] \hspace{1cm} (3.20)

With this choice of \( \alpha \) for all \( i \), the convergence factor satisfies,

\[ |\lambda_i| = |1-\alpha u_i| < 1 - \frac{2}{a+b} \cdot a = \frac{b-a}{b+a} \]

\[ = \frac{P-1}{P+1} \]
where \( P = b/a = \frac{\max \mu_i}{\min \mu_i} \) and is known as the \( P \)-condition number of matrix \( A \).

There is also another condition number known as the \( K \)-condition number and is defined as,

\[
K = (P)^{\frac{1}{2}} = \left( \frac{\max \mu_i}{\min \mu_i} \right)^{\frac{1}{2}}
\]

Westlake (1968).

In this chapter only the \( P \)-condition number is used, however in Chapter 4 the \( K \)-condition number is considered.

Earlier in this chapter the rate of convergence \( R \) was defined as

\[
R = -\log_e (|\lambda_1|)
\]

where \( |\lambda_1| \) is the spectral radius of the error operator \((I-aA)\).

It is clear from the above that the rate of convergence

\[
R = \frac{2}{P}
\]

for large \( P \).

3.3 NON-STATIONARY ITERATIVE METHODS

Take as an example the general iterative scheme

\[
u^{(n+1)} = a(n) + f_n(u^{(n)})\]

If either \( a \) or \( f \) or both are dependent on \( n \) then the iteration process is said to be non-stationary. An example of such a process is the previously mentioned Richardson's method, i.e.,

\[
\begin{align*}
u^{(n+1)} &= u^{(n)} + \frac{\alpha}{n} (b-Au^{(n)})
\end{align*}
\]

(3.21)

The \( \alpha_n \) may be determined by either,

a) a formulae involving \( u^{(n)} \) - this is done at each iteration

or b) in advance from known properties of the matrix \( A \).

The error vector for the scheme (3.21) can be shown to satisfy

\[
\begin{align*}
e^{(n+1)} &= (I-\alpha_n A)e^{(n)} \\
&= \prod_{i=0}^{n} (I-\alpha_i A)e^{(0)}
\end{align*}
\]

(3.22)
This clearly shows that the method is non-stationary as the error operator is not constant throughout the iterative process and changes for each iteration.

The $a$'s for use in Richardson's method may be obtained by using the following formula

$$\alpha_i^{(m)} = \frac{2}{(b+a)-(b-a)\cos\left[\frac{(2i-1)}{2m}\right]}$$

where $b$ and $a$ are the largest and smallest eigenvalues of $A$ respectively. The derivation of this formula has been omitted but may be found in Young (1954a).

A reasonable value of $m$ is chosen and the $a$'s thus obtained are used in a cyclic order such as

$$\alpha_i^{(m)} = \alpha_j$$

where $j$ is an integer such that $j \equiv i \pmod{m}$ and $1 \leq j \leq m$.

A better method which has been more successful, is to use the ordering

$$\alpha^{(m)}_1, \alpha^{(m+1)}_m, \alpha^{(m+1)}_{m-1}, \ldots, \alpha^{(m)}_{(m+1)/2}, \alpha^{(m)}_{(m+1)/2+1}, \ldots, \alpha^{(m)}_{(m+1)/2-1},$$

where $m$ is even, and

$$\alpha^{(m)}_1, \alpha^{(m+1)}_m, \alpha^{(m+1)}_{m-1}, \ldots, \alpha^{(m)}_{(m+1)/2}, \alpha^{(m)}_{(m+1)/2+1}, \ldots, \alpha^{(m)}_{(m+1)/2-1},$$

where $m$ is odd. Here we assume that the $\alpha_i^{(m)}$ are labelled with descending magnitude, i.e. $|\alpha_1^{(m)}| \geq |\alpha_2^{(m)}| \geq \ldots \geq |\alpha_m^{(m)}|$. This procedure has the advantage of using alternately large and small $\alpha_i^{(m)}$, thus preventing uninterrupted growth of round-off errors. [Young (1954a)].

### 3.4 SECOND ORDER ITERATIVE METHODS

Consider the second order Richardson's method

$$u^{(n+1)} = u^{(n)} + \alpha (b-Au^{(n)}) + \beta (u^{(n)} - u^{(n-1)})$$

(3.24)

To find $u^{(n+1)}$, knowledge of two previous iterates, $u^{(n)}$ and $u^{(n-1)}$, is required.
The error vector for this scheme can be shown to satisfy
\[ \mathbf{e}^{(n+1)} = [(1+\beta)I-\alpha\mathbf{A}]\mathbf{e}^{(n)} - \beta\mathbf{e}^{(n-1)} \quad (3.25) \]
This error operator is constant throughout the iteration and so the scheme is stationary.

Values of \(\alpha\) and \(\beta\) may be obtained from the following formulae
\[ \alpha = \frac{4}{\mu_1^2 + \mu_N^2} \quad , \quad \beta = \left(\frac{\sqrt{\mu_N} - \sqrt{\mu_1}}{\sqrt{\mu_N} + \sqrt{\mu_1}}\right)^2 \quad (3.26) \]
where \(\mu_1\) and \(\mu_N\) are the largest and smallest eigenvalues respectively of \(\mathbf{A}\).

The rate of convergence can be shown to be \(2/\sqrt{P}\) where \(P\) is the \(P\)-condition number of matrix \(\mathbf{A}\). This second order method thus converges faster than the first order method considered earlier in this chapter. However at least two vectors need to be stored for each iteration and this storage requirement can become quite prohibitive for large systems of equations especially on small computers when first order methods might prove more desirable.

An example of a non-stationary second order iterative method is the Chebyshev method defined by
\[ \mathbf{u}^{(n+1)} = \mathbf{u}^{(n)} + \alpha_n (\mathbf{b} - \mathbf{A}\mathbf{u}^{(n)}) + \beta_n (\mathbf{u}^{(n)} - \mathbf{u}^{(n-1)}) \quad (3.27) \]
where \(\mathbf{A}\) is Hermitian and positive definite and \(\alpha_n\) and \(\beta_n\) vary with each iteration.

The values of the parameters \(\alpha_n, \beta_n\) are defined by Steifel (1958) to be of the form,
\[ \alpha_n = \frac{4T_n(y)}{(b-a)T_{n+1}(y_0)} \quad , \quad \beta_n = \frac{T_{n-1}(y_0)}{T_{n+1}(y_0)} \quad (3.28) \]
where \(y_0 = \frac{b+a}{b-a} > 1\)

and \(T_r(x) = \cos(r\cos^{-1}x)\) is the Chebyshev polynomial of degree \(r\), adjusted to the range \([-1,1]\). The quantities \(b\) and \(a\) are respectively the largest and smallest eigenvalues of the matrix \(\mathbf{A}\).
Note that the recurrence relationship
\[ T_{r+1}(x) = 2xT_r(x) - T_{r-1}(x), \text{ for } r=2, \ldots, n-1 \text{ with } \]
\[ T_0(x) = 1, \ T_1(x) = x \]
is used to evaluate the function \( T_n(y_0) \) in (3.28).

3.5 SUCCESSIVE ITERATIVE METHODS

It was stated earlier in this chapter that if the matrix \( E \) in (3.4) is chosen as a lower or upper triangular matrix then this leads to methods of successive displacements.

If \( A=I-L-U \), where \( L,U \) are lower and upper, triangular matrices as defined earlier, then if \( E=(I-L) \) and \( F=U \) equation (3.4) can be written as,
\[ u(n+1) = Lu(n+1) + Uu(n) + b \]  
(3.30)
This is known as the Gauss-Seidel method of iteration. Re-writing (3.30) gives,
\[ u(n+1) = u(n) + L\bar{u}(n+1) + U\bar{u}(n) + b - u(n) \]  
(3.31)

In the same way as with the simultaneous methods an acceleration parameter \( \omega \) can be introduced into equation (3.31). This leads to a method known as the Successive Over-Relaxation or S.O.R. method,
\[ u(n+1) = u(n) + \omega [L\bar{u}(n+1) + U\bar{u}(n) + b - u(n)] \]  
(3.32)
Young (1954a) has shown that this method for solving equations such as (3.1) converges for \( 0<\omega<2 \) if \( A \) is a symmetric positive definite matrix. Also the optimum value, \( \omega_b \), of \( \omega \) for maximum rate of convergence is given by
\[ \omega_b = \frac{2}{1+(1-\bar{u}^2)^{\frac{1}{2}}} \]  
(3.33)
where \( \bar{u} \) is the spectral radius of the Jacobi iteration matrix \( (L+U) \).

If \( E=(I-U) \) and \( F=L \) is substituted into equation (3.4) then a different form of the Gauss-Seidel and the SOR methods are obtained,
\[ u(n+1) = Lu(n) + Uu(n+1) + b - \text{ Gauss-Seidel} \]  
(3.34)
\[ u^{(n+1)} = u^{(n)} + \omega [L_u^{(n)} + U_u^{(n+1)} + b - u^{(n)}] - S.O.R. \] (3.35)

The same conditions for convergence hold true for (3.35) as did for (3.32).

Another successive iterative method is known as the SSOR method and is a modification of the SOR method (3.31) wherein each iteration consists of two half iterations - a forward iteration followed by a backward iteration. The forward iteration is simply the SOR method (3.31), while the backward iteration is the (backwards) SOR method where the equations are taken in reverse order. If \( u^{(n+1)} \) represents the state of \( u \) after the forward sweep of the \( (n+1) \)th iteration has been performed then the SSOR method can be written as,

\[
\begin{align*}
\underline{u}^{(n+1)} &= \underline{u}^{(n)} + \omega [L_u^{(n+1)} + U_u^{(n)} + b - u^{(n)}] \\
\underline{u}^{(n+1)} &= \underline{u}^{(n+1)} + \omega [L_u^{(n+1)} + U_u^{(n+1)} + b - u^{(n+1)}]
\end{align*}
\] (3.36)

The SSOR method is not as efficient as the SOR method. However, because the eigenvalues of the coefficient matrix obtained by the SSOR method are real this allows acceleration methods to be applied more easily, [Jennings (1977)].

In discussions on the SSOR method Axelsson (1974) has proposed a formula to give the optimal acceleration parameter in the form,

\[ \omega = \frac{2}{1 + \sqrt{1 + ch}} \] (3.37)

where \( c \) is a constant and \( h \) the mesh size.
CHAPTER 4

THE USE OF PRECONDITIONING IN SOLVING

A TOEPLITZ SYSTEM OF SIMULTANEOUS EQUATIONS
4.0 INTRODUCTION

In the last chapter, it was shown that for both Richardson's method and the method of simultaneous displacements the optimum rate of convergence depended inversely upon the value of \( P \), the \( P \)-condition number of the iteration matrix. Hence if the system of linear equations can be "pre-conditioned" in some way so that the \( P \)-condition number is minimised then there will be a corresponding optimisation of the iterative method.

Again, using the same system as in §3.0, then

\[ Au = b, \quad (4.1) \]

where \( A \) is a symmetric, positive definite matrix and \( A = I - L - U \), with \( L, U \) being lower and upper triangular matrices, respectively, with zero diagonal elements.

Pre-multiplying equation (4.1) by \((I - \omega L)^{-1}\) gives

\[ (I - \omega L)^{-1}A(I - \omega U)^{-1}(I - \omega U)u = (I - \omega L)^{-1}b. \quad (4.2) \]

Letting,

\[ y = (I - \omega U)u, \quad G = (I - \omega U)^{-1}, \]

and

\[ d = (I - \omega L)^{-1}b \]

then (4.2) can then be re-written as

\[ G^T AGy = d \]

or

\[ By = d \]

(4.3)

It should be noted that the matrix \( G \) is in a form which is readily inverted, i.e. the evaluation of \( z = Gy \) where \( y \) is a known vector can be obtained by the solution of sets of equations like \((I - \omega U)z = y\) which, because of the form of the matrix is simply a back-substitution process. Similarly, the evaluation of \( z = G^T y \) is a forward substitution process which involves only the terms in the lower triangular part of matrix \( A \). The sparseness of \( A \) is therefore retained in \((G^T)^{-1}\) and \((G)^{-1}\) as these are simply the lower and upper triangular components of the original matrix \( A \).
Since $G$ is a non-singular matrix and $A$ is positive definite and symmetric, it therefore possesses a positive definite square root matrix $A^\frac{1}{2}$ which is also symmetric. Hence,

$$G^T A G = (G^\frac{1}{2} A^\frac{1}{2}) (A^\frac{1}{2} G)$$

(4.4)

from which it follows that $G^T A G$ is a positive definite symmetric matrix.

The optimum acceleration parameter $\bar{\omega}$ is now defined as that value of $\omega$ for which the ratio of the largest to the smallest eigenvalue $\lambda_1(\omega)/\lambda_N(\omega)$ of the matrix $G^T A G$ is minimised. Evans (1973) has shown that

$$\bar{\omega} = \frac{(\tau_N - \tau_1)}{(\kappa_1 - \kappa_N) - (\tau_N - \tau_1)(\kappa_N \tau_1 - \kappa_1 \tau_N - \kappa_1 \kappa_N \tau_1)}$$

(4.5)

where $\tau_i = v_i^T A v_i$, $\kappa_i = v_i^T L U v_i$ and $v_i$ are the eigenvectors of $(I - \omega U)^{-1}(I - \omega L)^{-1} A$.

Using this optimum acceleration parameter, $\bar{\omega}$, the pre-conditioned system can now be solved by some iterative method. A reduction in the number of iterations is obtained by using this pre-conditioned technique as shown in Evans (1973).

4.1 THE PRECONDITIONING OF A TOEPLITZ MATRIX

In this section the pre-conditioning technique is applied to the system

$$A u = b$$

(4.6)

where $A = I - L + U$ and $-L^T U$ i.e. the matrix $A$ is unsymmetric, $L$ and $U$ are again lower and upper triangular matrices with zero diagonal elements respectively.

As a particular case, consider the matrix $A$ as a Toeplitz matrix of the form,

$${}$$

(4.7)

$$
\begin{bmatrix}
1 & y & 0 \\
-y & 1 & y \\
0 & -y & 1
\end{bmatrix}
$$

*The motivation for the use of (4.7) can be found in §4.4.*
so that here $L^T = U$ if $A = I - L + U$.

Pre-multiplying (4.6) by $(I - \omega L)^{-1}$ gives,

$$(I - \omega L)^{-1}A(I + \omega U)^{-1}(I + \omega U)u = (I - \omega L)^{-1}b$$

Let $x = (I + \omega U)u$, $F = (I + \omega U)^{-1}$, $H = (I - \omega L)^{-1}$ and $c = (I - \omega L)^{-1}b$.

Equation (4.7) can now be re-written as

$$HAFx = c$$

or

$$Qx = c$$

(4.9)

As in §4.0, the inverse of the matrices $F$ and $H$ with respect to a right hand side can still be obtained by backward and forward substitution processes respectively. $F^{-1}$ and $H^{-1}$ are simply upper and lower triangular sections of the coefficient matrix $A$ so again, the sparseness of $A$ is preserved.

The optimum acceleration parameter $\omega$ is defined as that value of $\omega$ for which a condition number of $Q$ is minimised. In the last chapter, the $P$-condition number of matrix $Q$ was specified as

$$P(Q) = \frac{|\lambda|_{\text{max}}}{|\lambda|_{\text{min}}}$$

(4.10)

where $|\lambda|_{\text{max}}$ and $|\lambda|_{\text{min}}$ are the largest and smallest eigenvalues of $Q$ respectively. However, since $Q$ is unsymmetric it may have complex eigenvalues [Forsythe (1967)], and so a different condition number might be used in this case. This is defined by Forsythe (1967) as:

$$\text{condition number } (Q) = (\frac{\mu_1}{\mu_N})^\frac{1}{2} \geq 1$$

(4.11)

where $\mu_1, \mu_N$ are the largest and smallest eigenvalues of the symmetric matrix $Q^T Q$.

Hence,

$$\text{condition number } (Q) = K \text{ condition number } (Q^T Q)$$

(4.12)

the $K$-condition number having been defined in §3.2.

The question immediately arises as to which condition number should be
used to find the optimum acceleration parameter $\bar{w}$. Should $P(Q)$ or $K(Q^TQ)$ be minimised? This problem is overcome by using both formulae and comparing experimental results.

First the optimum acceleration parameter $\bar{w}$ is defined as that value of $\omega$ for which the $P$ condition number of $Q$ is minimised. If $Q$ has eigenvalues $\lambda_i(\omega)$ then the $P$ condition number for a particular value of $\omega$ would be $|\lambda_1(\omega)/\lambda_N(\omega)|$, where $\lambda_1(\omega)$ and $\lambda_N(\omega)$ are the eigenvalues of maximum and minimum modulus of the matrix $Q$ respectively.

Let $T=(I+\omega U)^{-1}(I-\omega L)^{-1}A$ where $T$ has eigenvectors $s_i$ with corresponding eigenvalues $\eta_i(\omega)$.

Therefore,

$$
(I-\omega L)^{-1}A s_i = \eta_i(I+\omega U)s_i \tag{4.13}
$$

By putting, $v_i = (I+\omega U)s_i$, then it follows that

$$(I-\omega L)^{-1}A(I+\omega U)^{-1}v_i = \eta_i v_i$$

$T$ has therefore the same eigenvalues as the matrix $Q$ but different eigen-vectors.

Equation (4.13) can now be written as

$$(I+\omega U)^{-1}(I-\omega L)^{-1}A s_i = \lambda_i s_i \tag{4.14}$$

hence,

$$A s_i = \lambda_i (I-\omega L)(I+\omega U)s_i$$

and so,

$$s_i^T A s_i = \lambda_i (s_i^T[I-\omega(L-U)-\omega LU]s_i)$$

$$= \lambda_i (s_i^T[I-\omega(I-A)-\omega LU]s_i) \tag{4.15}$$

Thus if $s_i^T s_i = 1$, then (4.15) becomes

$$\lambda_i = \frac{s_i^T A s_i}{1-\omega s_i^T A s_i - \omega^2 s_i^T L U s_i}$$

$$= \frac{\tau_i}{1-\omega \tau_i - \omega^2 k_i} \tag{4.16}$$
where $\tau_i = S_i^T A S_i$ and $\kappa_i = S_i^T L U S_i$.

Now, the $P$-condition number of $Q$,

$$P(\omega) = \frac{\lambda_1(\omega)}{\lambda_N(\omega)} = \frac{\lambda_1(1-\omega^2 \tau N - \omega^2 \kappa N)}{\lambda_N(1-\omega^2 \tau 1 - \omega^2 \kappa 1)}$$

which must be minimised to obtain the fastest rate of convergence.

For $P(\omega)$ to possess a stationary value, $dP(\omega)/d\omega$ must be equal to zero.

Therefore, differentiating $P(\omega)$ in equation (4.17) gives,

$$\frac{\tau_1 \tau_N}{\lambda_N(1-\omega^2 \tau 1 - \omega^2 \kappa 1)} \left\{ (\tau_1 \kappa_N - \tau_N \kappa_1 + \tau_1 \kappa_1) \omega + 2(\kappa_1 - \kappa_N) \omega + (\tau_N - \tau_1) \right\} = 0$$

Equation (4.18) can be expressed as a quadratic equation viz.

$$c\omega^2 + d\omega + f = 0$$

where

$$c = \tau_N \kappa_1 - \tau_1 \kappa_N - \kappa_N \kappa_1$$
$$d = 2(\kappa_1 - \kappa_N)$$
$$f = \tau_N - \tau_1$$

which gives the optimum pre-conditioning parameter $\bar{\omega}$ as

$$\bar{\omega} = \frac{(\tau_N - \tau_1)}{(\kappa_N - \kappa_1) - \frac{1}{2} \left( (\kappa_1 - \kappa_N)^2 - (\tau_N - \tau_1) (\tau_N \kappa_N - \tau_1 \kappa_N + \kappa_1 \kappa_1) \right)}$$

This result is similar to that shown in §4.0 when the matrix $A$ was symmetric.

The minimum $P$-condition number $\bar{P}$ is then given by,

$$\bar{P} = \frac{\lambda_1(1+\bar{\omega}^2 \tau N - \bar{\omega}^2 \kappa N)}{\lambda_N(1+\bar{\omega}^2 \tau 1 - \bar{\omega}^2 \kappa 1)}$$

If the eigenvalue spectrum of the matrix $Q$ are real and are such that

$$0 < \bar{a} < \lambda_1(\omega) < \bar{b}, \ i=1,2,\ldots,N$$

then

$$\bar{a} \leq \frac{\tau_N}{(1-\bar{\omega}^2 \tau N - \bar{\omega}^2 \kappa N)}$$

$$\bar{b} \geq \frac{\tau_1}{(1-\bar{\omega}^2 \tau 1 - \bar{\omega}^2 \kappa 1)}$$
Thus, with some restrictions, expressions have been derived for the
eigenvalue spectrum, minimum P-condition number and the optimum pre-
conditioning parameter of $Q$ in terms of the quantities $T_l, T_N, \kappa_l$ and $\kappa_N$.

Now consider the second alternative where the optimum acceleration
parameter $\tilde{w}$ is defined as that value of $w$ for which the K-condition number
of $Q^T Q$ is minimised. Let $Q^T Q$ have eigenvalues $\mu_i$ and so the K-condition
number for a particular value of $w$ is $(\mu_1/\mu_N)^1$ where $\mu_1, \mu_N$ are the largest
and smallest eigenvalues of $Q^T Q$ respectively.

Now
\begin{equation}
Q = (I-\omega L)^{-1} A (I+\omega U)^{-1}
\end{equation}
giving,
\begin{equation}
Q^T Q = (I+\omega L)^{-1} A (I-\omega U)^{-1} (I-\omega L)^{-1} A (I+\omega U)^{-1}
\end{equation}

A formula, similar to (4.16) in which the eigenvalues of $Q^T Q$ are
expressed in terms of $A^T A$ and the eigenvectors of $Q^T Q$ is now sought. From
this a formula for the optimum acceleration parameter $\tilde{w}$ can be found.
However, expression (4.25) is very complicated and thwarted all attempts to
find transformations of $Q^T Q$ which would give formulae for the eigenvalues
in terms of quantities similar to $T_l, T_N, \kappa_l$ in (4.16). Experimental analysis
had to be resorted to, and results from this analysis are shown in §4.3.

4.2 THE PRECONDITIONED SIMULTANEOUS DISPLACEMENT ITERATIVE METHOD

In Chapter 3, the accelerated Jacobi or J.O.R. simultaneous iterative
method for solving the system $A\underline{x}=\underline{b}$, where $A$ is symmetric, was introduced.
If this method is applied to the preconditioned system (4.9) another method,
the preconditioned simultaneous displacement iterative method (4.26) is
formed,
\begin{equation}
\underline{x}^{(n+1)} = \underline{x}^{(n)} + \alpha [c - Q \underline{x}^{(n)}]
\end{equation}
This preconditioned iterative method can be used to solve the system (4.6)
$A\underline{x}=\underline{b}$, where $A$ is the Toeplitz matrix (4.7), by first preconditioning the
system, as shown in §4.1, to form a preconditioned system (4.9), \( Qx = c \). The solution \( x \) of this preconditioned system can then be found by using the method (4.26). However, vector \( u \), the solution of (4.6), is required. This can be obtained by 'deconditioning' the vector \( x \), which is simply a backward substitution process.

The above preconditioning process for a Toeplitz matrix (4.7) is laid out comprehensively in the following example.

**Example 4.1**

Solve

\[
Au = b
\]

where \( A \) is an \( N \times N \) Toeplitz matrix of the form (4.7). Therefore,

\[
(I + \omega U) = \begin{bmatrix}
1 & \omega y & 0 \\
\omega y & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

(4.28)

and

\[
(I - \omega L) = \begin{bmatrix}
1 & -\omega y & 0 \\
-\omega y & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

(4.29)

The following process, steps 1-8, illustrates the usage of the preconditioned simultaneous displacement iterative method (4.26) in solving the system (4.27).

**STEP 1:** Find \( c \)

Now \( c = (I - \omega L)^{-1}b \)

\( \Rightarrow \) \( b = (I - \omega L)c \)

Hence, \( c \) can be found by the forward substitution process,
\[ c_1 = b_1 \]
\[ c_i = b_1 + w y c_{i-1}, \quad i = 2, 3, \ldots, N. \]

**STEP 2:** Find \( x^{(0)} \) after making an initial guess of \( u \)
\[ x^{(0)} = (I+wU)u. \]

This is simply a forward substitution process
\[ x_i = u_i + wy u_{i+1}, \quad i = 1, 2, \ldots, (N-1), \]
\[ x_N = u_N \]

**STEP 3:** Find \( (I+wU)^{-1} x^{(n)} \)

Let \( x = (I+wU)^{-1} x^{(n)} \)
\[ \Rightarrow x^{(n)} = (I+wU)x \]

Consequently, \( z \) can be obtained by a simple backward substitution process.
\[ z_N = x_N^{(n)} \]
\[ z_i = x_i^{(n)} - w y z_{i+1}, \quad i = N-1, N-2, \ldots, 1. \]

**STEP 4:** Obtain \( A z \)
\[ z_1 = A_{1,1} z_1 + A_{1,2} z_2 \]
\[ z_i = A_{i,i} z_i + A_{i,i-1} z_{i-1} + A_{i,i+1} z_{i+1}, \quad i = 2, 3, \ldots, N-1 \]
\[ z_N = A_{N,N} z_N + A_{N,N-1} z_{N-1} \]

**STEP 5:** Finally \( (I-\omega L)^{-1} z \) is found by, a forward substitution process
\[ \omega_1 = z_1 \]
\[ \omega_i = z_i + w y z_{i-1}, \quad i = 2, 3, \ldots, N \]

**STEP 6:** Find \( x^{(n+1)} \)

At this point the original system (4.27) has been converted into the preconditioned system (4.9) and so now the preconditioned simultaneous displacement iterative method for example, can be used, i.e.
\[ x^{(n+1)} = x^{(n)} + a(c-\omega) \]
STEP 7: Check that the difference $x^{(n+1)} - x^{(n)}$ is less than some pre-defined tolerance. If not then go back to Step 3 and repeat process.

STEP 8: Decondition $x^{(n+1)}$ to obtain the required solution $u$.

$$u_N = x^{(n+1)}_N$$

$$u_i = x^{(n+1)}_i - \omega y_{i,i+1} x^{(n+1)}_{i+1}, \quad i = N-1, \ldots, 2, 1.$$

Attention is brought to the fact that the inverse of $(I-\omega L)$ and $(I+\omega U)$ need never be found thus eliminating some tedious computation from this preconditioned iterative process.

### 4.3 EXPERIMENTAL ANALYSIS

In §4.1 the optimum acceleration parameter $\bar{\omega}$ for the preconditioned system (4.9) was defined in terms of two condition numbers, the P condition number of $Q$ and the K-condition number of $Q^TQ$. As the eigenvalues of $Q$ may be complex, the modulus is used, i.e. $\sqrt{(\text{real part})^2 + (\text{imaginary part})^2}$. Consider a 20x20 Toeplitz matrix of the form (4.7) with $\gamma = 1/4$. By preconditioning this matrix to form the matrix $Q$ (4.9) and finding $\bar{\omega}$ by the above mentioned methods, it can be seen from Figures 4.1 and 4.2 that the P-condition number method gives a different optimum acceleration parameter than the K-condition number method. If a value for $\bar{\omega}$ is obtained experimentally, i.e. by using (4.26) with $\alpha = 1$ to find an $\omega$, which gives the true solution to (4.6) in the least number of iterations, and compared with the values of $\bar{\omega}$ obtained from Figures 4.1 and 4.2 then it is clear that the $\bar{\omega}$ found by using the P-condition number is closer to the experimental result. (Table 4.1). It is also apparent from Table 4.1 that the order of the original Toeplitz coefficient matrix has no effect upon the value of $\bar{\omega}$ found experimentally or from the eigenvalues of the matrix $Q^TQ$. However, in the case of $\bar{\omega}$ obtained by the method employing the P-condition number of matrix $Q$ it can...
<table>
<thead>
<tr>
<th>ORDER</th>
<th>OFF DIAGONAL ELEMENT $y = 1/4$</th>
<th>OFF DIAGONAL ELEMENT $y = 1/2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EXPERIMENTAL</td>
<td>USING P COND(Q)</td>
</tr>
<tr>
<td>10</td>
<td>0.83</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$</td>
<td></td>
</tr>
<tr>
<td>0.88</td>
<td>0.83</td>
<td>0.82</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$</td>
<td></td>
</tr>
<tr>
<td>0.88</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td></td>
<td>$\downarrow$</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 4.1**

Optimum acceleration parameters for two different Toeplitz matrices of the form (4.7)
FIGURE 4.1: Eigenvalues and corresponding K condition number of $Q^TQ$, where the original Toeplitz matrix is of order 20 and has off diagonal elements of $1/4$. 
FIGURE 4.2: Modulus of eigenvalues and corresponding P condition number of Q, where original Toeplitz matrix is of order 20 and has off diagonal elements of 1/4.
be seen that the $\bar{w}$ so found moves slowly away from the experimental $\bar{w}$ as the order of the Toeplitz matrix increases. This deviation from the experimental $\bar{w}$ is however quite small and so the Toeplitz matrix would have to be very large, i.e., greater than $100 \times 100$, for the difference to be effectual. These conclusions are also true for Toeplitz matrices of the form (4.7) with values of $\gamma$ other than 1/4.

Up till now, the eigenvalues of $Q$ or $Q^TQ$ had to be found in order to estimate the optimum acceleration parameter $\bar{w}$. If some formula could be suggested which will give an approximate $\bar{w}$, without having to find eigenvalues of $Q$ or $Q^TQ$, then most of the work involved in finding $\bar{w}$ would be alleviated.

A tentative analysis suggests that an estimated value of $\omega$ can be obtained from,

$$\bar{w} = \frac{2}{1+(1+4\gamma)^{1/2}} \quad (4.30)$$

where $\gamma$ is the off-diagonal element of the Toeplitz matrix. From Table 4.2 it can be seen that values of $\bar{w}$ found by (4.31) are quite near to those obtained experimentally.

<table>
<thead>
<tr>
<th>OFF DIAGONAL ELT. $\gamma$</th>
<th>BY FORMULA</th>
<th>BY EXPERIMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/6</td>
<td>0.87</td>
<td>0.7 + 1.2</td>
</tr>
<tr>
<td>1/4</td>
<td>0.83</td>
<td>0.83 + 0.88</td>
</tr>
<tr>
<td>1/3</td>
<td>0.79</td>
<td>0.79 + 0.9</td>
</tr>
<tr>
<td>1/2</td>
<td>0.73</td>
<td>0.75 + 0.759</td>
</tr>
<tr>
<td>3/5</td>
<td>0.70</td>
<td>0.69 + 0.76</td>
</tr>
</tbody>
</table>

**TABLE 4.2**

Values of $\bar{w}$ obtained by (4.30) and by experiment.

The RHS vector is such that the solution of the equation is unity. Comparing the preconditioned simultaneous displacement iterative method with other well known methods, as is done in Table 4.3, it is obvious that there is a saving on the number of iterations when the preconditioned method is used.

Since the preconditioned method can be considered as a variant of the SSOR method, Axelsson (1974), then it seems appropriate to choose a formula similar to (3.37) which can predict $\bar{w}$. By trial and error (4.30) was found to be the most suitable.
<table>
<thead>
<tr>
<th>METHOD</th>
<th>10×10 MATRIX</th>
<th>20×20 MATRIX</th>
<th>40×40 MATRIX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ACCELERATION</td>
<td>LEAST NO.</td>
<td>ACCELERATION</td>
</tr>
<tr>
<td></td>
<td>PARAMETER</td>
<td>OF ITERATIONS</td>
<td>PARAMETER</td>
</tr>
<tr>
<td>J.O.R.</td>
<td>0.8</td>
<td>12</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAUSS SEIDEL 1</td>
<td>-</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GAUSS SEIDEL 2</td>
<td>-</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.O.R. 1</td>
<td>0.9</td>
<td>9</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.O.R. 2</td>
<td>0.9</td>
<td>10</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRECONDITIONED SIM.DISP.</td>
<td>0.83</td>
<td>5</td>
<td>0.83</td>
</tr>
<tr>
<td>(α=1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.88</td>
<td></td>
<td>0.88</td>
</tr>
</tbody>
</table>

**TABLE 4.3**

<table>
<thead>
<tr>
<th>Toeplitz matrix with off-diagonal element of 1/4</th>
</tr>
</thead>
</table>

The RHS vector is such that the solution of the equation is unity.
The convergence criteria used in all the popular iterative methods was,

\[
\frac{|u^{(n+1)} - u^{(n)}|}{1 + |u^{(n)}|} < 5 \times 10^{-6}
\]

and similarly for the preconditioned iterative method.

The results shown in Table 4.3 for the preconditioned method have all been obtained with \(\alpha=1\). However, when the optimum value of \(\alpha\) is used there is even more saving on the number of iterations.

Preconditioned method for a Toeplitz matrix order 20

<table>
<thead>
<tr>
<th>OFF DIAGONAL ELT. (y)</th>
<th>OPTIMUM (\omega)</th>
<th>(\alpha)</th>
<th>NO. OF ITERATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/6</td>
<td>0.69, 1.22</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>0.9, 0.97, 0.99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/4</td>
<td>0.83, 0.88</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>0.85, 0.92, 0.99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/3</td>
<td>0.79, 0.90</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0.85, 0.92, 0.96</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1/2</td>
<td>0.75, 0.759</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>0.75, 0.84, 0.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3/5</td>
<td>0.69, 0.77</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>0.73, 0.83, 0.86</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(*): Optimum value of \(\alpha\), found by keeping \(\omega\) constant and varying \(\alpha\).

The RHS vector is such that the solution of the equation is unity. Having established that the preconditioned method converges in fewer iterations than the established methods considered, it is now necessary to
calculate the total number of operations per iteration. From this, the
total work (no. of operations per iteration × no. of iterations) for each
of the iterative schemes to converge can be obtained, thus enabling a
comparison of the 'speed' of the different techniques to be made. To
simplify this comparative process, the preconditioned iterative process
(4.26) is matched with one of the well known iterative techniques - the
accelerated Jacobi or J.O.R. method (3.14).

It is assumed that, on an average computer a multiplication takes twice
as long to execute as an addition hence,

\[ \text{One operation} = \text{One multiplication} = 2 \text{ additions} \] \hspace{1cm} (4.31)

The preconditioned simultaneous displacement iterative method requires
\((16N-12)\) operations per iteration where \(N\) is the order of the original
Toeplitz matrix. In addition, there is an overhead of \((9N-5)\) operations.
On the other hand, the J.O.R. iterative method requires only \((10N-6)\)
operations per iteration. With these formulae for the number of operations
per iteration, a table can be formed showing the total number of operations
taken by each of the two methods to converge to the true solution produced.

<table>
<thead>
<tr>
<th>OFF DIAGONAL ELT. OF TOEPLITZ MATRIX</th>
<th>ORDER</th>
<th>PRECONDITIONED METHOD</th>
<th>J.O.R. METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td>1407</td>
<td>1746</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>5787</td>
<td>7146</td>
</tr>
<tr>
<td>1/6</td>
<td>20</td>
<td>1715</td>
<td>2135</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>7055</td>
<td>8734</td>
</tr>
<tr>
<td>1/4</td>
<td>20</td>
<td>2023</td>
<td>4462</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>8323</td>
<td>18262</td>
</tr>
<tr>
<td>1/2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 4.5**

*Number of operations for Toeplitz system to converge*

The RHS vector is such that the solution of the equation is unity.

It can plainly be seen from this table that the preconditioned method
requires less operations than the J.O.R. method in all the cases considered.
4.4 MOTIVATION FOR THE PRECONDITIONING OF A TOEPLITZ MATRIX

Consider the two dimensional elliptic difference equation,

$$\varepsilon \left( \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} \right) + \frac{3}{\varepsilon} \frac{\partial U}{\partial x} + \frac{3}{\varepsilon} \frac{\partial U}{\partial y} + U = 0$$

(4.32)

for \((x,y) \in \mathbb{R}; U=0\) for \((x,y) \in \mathbb{C}\).

Let the domain \(\mathbb{R}\) be a unit square with sides parallel to the coordinate axes with a mesh size of \(h\). Now, if the derivatives in (4.32) are replaced by the finite difference formulae derived in Chapter 1, the following elliptic difference equation is obtained,

$$\left[ 1 - \frac{4\varepsilon}{h^2} \right] u_{i,j} + \left[ 1 + \frac{2\varepsilon h^{-1}}{2h} \right] \left[ u_{i+1,j} + u_{i,j+1} \right] + \left[ 1 + \frac{2\varepsilon h^{-1}}{2h} \right] \left[ u_{i-1,j} + u_{i,j-1} \right] = 0$$

(4.33)

as an approximation to (4.32) at the mesh point \((ih,jh)\). If this formula is applied to the row-wise ordering of points of the square for a small value of \(\varepsilon\), say of order \(10^{-5}\); then a matrix equation of the form,

$$Au = b$$

(4.34)

is obtained. \(A\) is the large sparse block Toeplitz matrix,

$$\begin{bmatrix}
  x & y_1 & y_2 \\
  -y_1 & x & y_2 \\
  0 & -y_1 & x
\end{bmatrix}$$

(4.35)

where

$$x = \begin{bmatrix}
  1 & 1/2h \\
  -1/2h & 1 & 1/2h \\
  0 & -1/2h & 1
\end{bmatrix}$$

and

$$y_1 = y_2 = \begin{bmatrix}
  1/2h \\
  0 \\
  1/2h
\end{bmatrix}$$

It is desirable to solve (4.35) by a preconditioned iterative method as the direct method specified by Evans (1980) is suitable only for one dimensional problems. Since knowledge of the preconditioning theory of
unsymmetric matrices is minimal then a simpler matrix needs to be considered first before any preconditioning theory can be applied to (4.35).

The system which is investigated is obtained by solving the singular perturbation problem, i.e. the 2 point boundary value problem given by the differential equation,

$$
e \frac{d^2 u}{dx^2} + \frac{du}{dx} + U = 0$$  \hspace{1cm} (4.36)

on the interval (0,1) with boundary conditions \( u(0)=a \) and \( u(1)=b \). When discretised this would produce a set of difference equations with a coefficient matrix of the form,

$$
\begin{bmatrix}
1 - \frac{2\epsilon}{h^2} & \frac{1+2\epsilon h^{-1}}{2h} \\
\frac{-1+2\epsilon h^{-1}}{2h} & 1 - \frac{2\epsilon}{h^2} & \frac{1+2\epsilon h^{-1}}{2h} \\
0 & \frac{-1+2\epsilon h^{-1}}{2h} & 1 - \frac{2\epsilon}{h^2}
\end{bmatrix}, \hspace{1cm} (4.37)
$$

In the limit as \( \epsilon \to 0 \) then (4.37) becomes,

$$
\begin{bmatrix}
1 & y \\
-y & 1 & y \\
0 & -y & 1
\end{bmatrix}, \hspace{1cm} (4.38)
$$

where \( y = \frac{1}{2h} \).

The preconditioning analysis for (4.38) is now investigated with the purpose of predicting results for the two dimensional case.
CHAPTER 5

THE SPARSE PRODUCT FORM OF THE INVERSE

AND ITS USE IN ITERATIVE METHODS
5.0 INTRODUCTION

The product form of the inverse or PFI was shown, in Chapter 2, to be one method of obtaining the inverse of a matrix (in product form) while carrying out the Gauss-Jordan elimination process on that matrix. In this chapter, a new concept, the sparse product form of the inverse or SPFI is introduced and used to obtain an approximate inverse of a sparse matrix, with the purpose of introducing the SPFI into several first and second order stationary/non-stationary iterative methods to form a preconditioned semi-iterative method. Finally, the method is compared with some well known and established iterative methods.

5.1 THE SPARSE PRODUCT FORM OF THE INVERSE

In §2.3 the product form of the inverse or PFI for a general N×N matrix A was introduced. This PFI is presented here again for convenience, with all the terms being simplified. Hence, in the PFI given below,

\[ x_{1,i} = -a_{1,i} / a_{i,i}, \quad (5.1) \]

i.e. the first row, i\textsuperscript{th} column of the PFI given in §2.3

\[
\begin{bmatrix}
  x_{11} & x_{12} & \cdots & x_{1i} & \cdots & x_{1N} \\
  x_{21} & x_{22} & \cdots & x_{2i} & \cdots & x_{2N} \\
  \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
  \vdots & \vdots & \ddots & x_{ii} & \ddots & \vdots \\
  x_{N1} & x_{N2} & \cdots & x_{Ni} & \cdots & x_{NN}
\end{bmatrix}
\]

PFI(A) =

If not all the terms \( x_{ij} \) are determined, but only those on a certain super-diagonal and below (all other elements being set to zero), then the result would be the sparse product form of the inverse, (SPFI).

Suppose that for illustration, A is a 6×6 matrix. Then the array,
would be SPFI(A) with a sparsity factor (SF) of 3.

Clearly, \( SPFI(A)_{SF=6} = PFI(A) \) (5.4)

An approximation to the inverse of matrix A can thus be formed by using the SPFI in the same manner as the PFI was used in §2.3 to find the true inverse of a matrix.

The advantage in using the SPFI as opposed to the PFI is that fewer elements need to be calculated which means a smaller number of operations. This can be seen in example 5.1.

It was observed that for the sparse positive definite systems \( Au=b \) under investigation, the fill in terms in the right upper triangle of the PFI of A were monotonically decreasing and so could be discarded since it is planned to use the approximate inverse of the matrix A so obtained in an iterative algorithm, [Dubois et al (1979)].

Example 5.1

In this example Gauss-Jordan elimination is performed on the 6×6 matrix A, shown below. The reader may recognise this matrix as being that which is obtained by applying the Poisson five point approximation formula (1.19) to the mesh points of the rectangular region in Figure 5.1 using a square mesh.

This example is presented only as an illustration of how the SPFI might be obtained. The actual method which is used can be found in §5.2. Sparsity factors of 1 and 6 are chosen as these provide the reader with a greater appreciation of the difference between the PFI and SPFI than say, a choice of 3 and 6.
The SPFI is obtained using a sparsity factor of 6 and 1. Note that when $SF=6$ the SPFI obtained is the same as the PFI for the matrix.

**STEP 1:**

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

**SPFI**

\[
\begin{bmatrix}
1/4 \\
1/4 \\
. \\
. 
\end{bmatrix}
\]

**SF=6**

\[
\begin{bmatrix}
1/4 \\
1/4 \\
. \\
. 
\end{bmatrix}
\]

**SF=1**

**STEP 2:** Eliminate column 1 to give

\[
\begin{bmatrix}
1 & -1/4 & -1/4 & -1/4 \\
. & 15/4 & -1 & -1/4 \\
. & -1 & 4 & . \\
. & -1/4 & 15/4 & -1 \\
. & -1 & -1 & 4 \\
. & . & -1 & -1 
\end{bmatrix}
\]

**SPFI**

\[
\begin{bmatrix}
1/15 \\
4/15 \\
4/15 \\
1/15 \\
4/15 \\
. 
\end{bmatrix}
\]

**SF=6**

\[
\begin{bmatrix}
1/15 \\
4/15 \\
4/15 \\
1/15 \\
4/15 \\
. 
\end{bmatrix}
\]

**SF=1**
STEP 3: Eliminate column 2 to give

\[
\begin{array}{cccc}
1 & -1/15 & -4/15 & -1/15 \\
1 & -4/15 & -1/15 & -4/15 \\
56/15 & -1/15 & -4/15 & -1 \\
-1/15 & 56/15 & -16/15 & . \\
-4/15 & -16/15 & 56/15 & -1 \\
-1 & . & -1 & 4 \\
\end{array}
\]

\[
\begin{array}{c}
1/56 \\
1/14 \\
15/56 \\
1/56 \\
1/14 \\
15/56 \\
\end{array}
\]

STEP 4: Eliminate column 3 to give

\[
\begin{array}{cccc}
1 & . & -15/56 & -1/14 \\
1 & -1/14 & -2/7 & -1/14 \\
1 & -1/56 & -1/14 & -15/56 \\
209/56 & -15/14 & -1/56 & 56/209 \\
-15/14 & 26/7 & -15/14 & 60/209 \\
-1/56 & -15/14 & 209/56 & 1/209 \\
\end{array}
\]

\[
\begin{array}{c}
15/209 \\
4/209 \\
1209/56 \\
56/209 \\
60/209 \\
1/209 \\
\end{array}
\]

STEP 5: Eliminate column 4 to give

\[
\begin{array}{cccc}
1 & . & -31/209 & -4/209 \\
1 & -64/209 & -15/209 & . \\
1 & -16/209 & -56/209 & 2/89 \\
1 & -60/209 & -1/209 & 15/178 \\
712/209 & -225/209 & . & 209/712 \\
-225/209 & 780/209 & . & 225/712 \\
\end{array}
\]

\[
\begin{array}{c}
31/712 \\
8/89 \\
2/89 \\
15/178 \\
209/712 \\
225/712 \\
\end{array}
\]

STEP 6: Eliminate column 5 to give

\[
\begin{array}{cccc}
1 & . & -47/712 & \text{..} \\
1 & -15/89 & . & \text{..} \\
1 & -26/89 & . & \text{..} \\
1 & -17/178 & . & \text{..} \\
1 & -225/712 & . & \text{..} \\
1 & 2415/712 & . & \text{..} \\
\end{array}
\]

\[
\begin{array}{c}
-47/2415 \\
-8/161 \\
208/2415 \\
-68/2415 \\
-15/161 \\
712/2415 \\
\end{array}
\]
In steps 3-6 the elements bounded by rectangles need not be obtained for SF=1, i.e. 20 less elements need to be calculated when the sparsity factor is 1 than when it has value 6. This is a saving of 35 multiplications and 20 additions in calculating the SPFI with a sparsity factor of 1. Table 5.1 shows the number of multiplications and additions saved in calculating the SPFI for matrix A (5.5) with different values of SF.

<table>
<thead>
<tr>
<th>SPARSITY FACTOR</th>
<th>MULTIPLICATIONS SAVED</th>
<th>ADDITIONS SAVED</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

TABLE 5.1

It can be seen from the above results that as the sparsity factor increases the saving becomes less until when SF=6 there is no saving. This is expected because the SPFI becomes closer to the PFI as SF gets bigger. It was explained earlier that when SF=6 then,

\[ \text{SPFI}(A) = \text{PFI}(A) \]

and so it is expected that there will be no operations saved in this case.

It follows from Example 5.1, that as the order of the matrix A increases then the greater will be the economy on the number of operations. However, the approximation of \( A^{-1} \) which can be obtained from the SPFI gets further and further away from the real inverse as the sparsity factor decreases. The problem is therefore to find a sparsity factor which produces the best approximation of the inverse of A using the least number of operations. This problem is tackled in §5.3.
5.2 AN ALGORITHM FOR FORMING THE SPFI

Sparse matrices of the same form as matrix A in Example 5.1, but with different bandwidth and order can be obtained by applying the Poisson five point difference approximation (1.19) to different mesh sizes (see § 1.2) within a square or rectangular region. This type of matrix is thus used to analyse the usefulness of somehow employing the SPFI concept in solving the Poisson equation (1.17) over a square or rectangular region. Hence, only this model matrix will be considered throughout this chapter.

An algorithm is given below which will produce the SPFI of the model N*N matrix, given a sparsity factor r.

The model matrix is held in five arrays as follows:

\[
\begin{bmatrix}
  b_1 & c_1 & x_1 & 0 \\
  a_2 & b_2 & c_2 & 0 & x_{N-q+1} \\
  a_3 & b_3 & c_3 & 0 \\
  y_1 & 0 & y_{N-q+1} & a_N & b_N \\
  0 & y_{N-q+1} & c_{N-1} & a_N & b_N
\end{bmatrix}
\]

and the SPFI is given in the array s. Account is taken of the fact that below the y band the values are all zero making the corresponding values in the SPFI also zero. Thus the number of operations in forming the SPFI is further reduced.

Find first column of SPFI

\[ s_{1,1} = \frac{1}{b_1} \]
\[ s_{2,1} = -\frac{a_2}{b_1} \]
\[ s_{q,1} = -\frac{y_1}{b_1} \]

Put ith col. of matrix A into vector v

\[ 1 \text{ for } i=2 \text{ to } N \text{ do} \]
2 for j=1 to N do
   v._=0
2 od
3 if i<>q then v._=q+1=x._-q+1 3 fi
4 if i<>N-q+1 then v._=y._-q+1 4 fi
v._=c._
v._=b._
5 if i<>N then v._=a._+1 5 fi

Multiply vector v by previous i-1 cols. of the SPFI

l=1
6 for j=1 to i-1 do
   7 if j<>1 then kl=q+j-1 else kl=N fi
   7 fi
   8 for k=1 to kl do
      9 if k>j-r then if j<>k
      then v._=v._+s._*v._
      else p=v._*s._
      fi
      9 fi
   8 od
   v._=p
6 od

Work out ith col. of SPFI

p=1/v._
10 if i<>i-1 then kl=q+i-1 else kl=N fi
10 fi
11 for j=1 to kl do
   12 if j>i-r then s._=v._*p
   12 fi
11 od
s._=p
1 od
5.3 **CHOOSING A SUITABLE SPARSITY FACTOR**

**Definition:**

The *Euclidean norm* of a vector $x$ is defined as,

$$||x||_E = \left[ \sum_{i=1}^{n} (x_i)^2 \right]^{1/2}$$

Consider again the model system which was derived in §1.2

$$Au = b$$

(S.6)

The solution of (S.6) can be obtained directly by finding the inverse of $A$ and applying it to,

$$u = A^{-1}b$$

(5.7)

The solution of (S.6) can also be found by using the product form of the inverse as shown in §2.3, i.e.

$$u = C^{(N)}(N-1),\ldots,(1)b)$$

(S.8)

If the $SPFI(A)^{SF=m}$ is used an approximate solution $u'$ will be found,

$$u' = D^{(N)}(N-1),\ldots,(1)b))$$

(S.9)

where $D(i)$ differs from a unit matrix only in the $i$th column. This column is the same as the $i$th column of the $SPFI$ when $SF=m$.

Let,

$$e = u' - u$$

(5.10)

i.e. the difference between the approximate solution and the real solution.

If the Euclidean norm of $e$ is found then this can be used as a measure of the 'closeness' of the approximate solution of (S.6) to the true solution. The absolute test (5.10) is used in preference to the relative test as, in all the cases considered the true solution was chosen to be unity, thus dispensing with the need for a relative test.

Table 5.2 displays the Euclidean norm of $e$ in (5.10) obtained by solving (S.6) by the SPFI method. The $N\times N$ matrix $A$ is blocked and has the form,
where \( B \) and \( C \) are \( M \times M \) matrices as follows

\[
B = \begin{bmatrix}
4 & -1 & \\
-1 & 4 & -1 & \\
-1 & 4 & -1 & 0 \\
0 & & & & \\
\end{bmatrix}
\]

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

and RHS vector \( b = \begin{bmatrix} 2 \\ 0 \\ . \\ . \\ 0 \\ 2 \end{bmatrix} \)

<table>
<thead>
<tr>
<th>SPARSITY FACTOR</th>
<th>BANDWIDTH 5</th>
<th>BANDWIDTH 11</th>
<th>BANDWIDTH 21</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.194</td>
<td>4.265</td>
<td>3.165</td>
</tr>
<tr>
<td>2</td>
<td>4.850</td>
<td>3.721</td>
<td>2.409</td>
</tr>
<tr>
<td>3</td>
<td>4.677</td>
<td>3.550</td>
<td>2.234</td>
</tr>
<tr>
<td>4</td>
<td>4.489</td>
<td>3.486</td>
<td>2.191</td>
</tr>
<tr>
<td>5</td>
<td>4.098</td>
<td>3.458</td>
<td>2.179</td>
</tr>
<tr>
<td>6</td>
<td>3.828</td>
<td>3.441</td>
<td>2.175</td>
</tr>
<tr>
<td>7</td>
<td>3.630</td>
<td>3.425</td>
<td>2.174</td>
</tr>
<tr>
<td>10</td>
<td>2.995</td>
<td>3.100</td>
<td>2.173</td>
</tr>
<tr>
<td>11</td>
<td>2.815</td>
<td>2.460</td>
<td>2.173</td>
</tr>
<tr>
<td>12</td>
<td>2.643</td>
<td>2.101</td>
<td>2.173</td>
</tr>
<tr>
<td>15</td>
<td>2.145</td>
<td>1.814</td>
<td>2.173</td>
</tr>
<tr>
<td>20</td>
<td>1.457</td>
<td>1.471</td>
<td>1.851</td>
</tr>
<tr>
<td>21</td>
<td>1.337</td>
<td>1.184</td>
<td>0.919</td>
</tr>
<tr>
<td>22</td>
<td>1.221</td>
<td>0.966</td>
<td>0.398</td>
</tr>
<tr>
<td>30</td>
<td>0.476</td>
<td>0.514</td>
<td>0.041</td>
</tr>
<tr>
<td>40</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

**TABLE 5.2:** Euclidean error norms
The matrix $A$ has bandwidth $(M+1)$ and is derived, as in Example 5.1, by applying the Poisson five point approximation formula (1.19) to the mesh points in a bounded rectangular region.

Figure 5.2 shows a plot of the values given in Table 5.2. Earlier, it was shown that, as the sparsity factor decreases, so does the number of operations required to find an approximate solution to the model problem (5.6). Consequently, a sparsity factor is required which gives a 'reasonable' value of the Euclidean norm, whilst at the same time keeping down the number of operations. From Figure 5.2 it can be seen that a sparsity factor of between 3 and 5 is a wise choice for bandwidths of 11 and 21 as the Euclidean norm is stable for sparsity factors of between 3 and 9 and at the same time has a moderate value. In scanning the plot of the 'bandwidth 5' case it is not immediately obvious as to the choice of sparsity factor. A choice of between 3 and 5 would produce an approximation of $A^{-1}$ which is too far away from the true inverse, but any larger sparsity factor would increase the number of operations needed to find a reasonable approximation. One way around this problem is to first, find an approximate inverse of $A$ by employing the SPFI method, and then obtain the true solution of the model system by some iterative method. A sparsity factor of between 3 and 5 can thus be used for the case when the bandwidth has a value 5.

Similar results to those shown in Table 5.2 have been obtained for larger and smaller orders of matrix $A$ but they have not been published in this report. They seem to suggest that, whatever the order or bandwidth of matrix $A$, an optimum choice for the sparsity factor is within the range 3 to 5.

N.B.: This is only true for Poisson's equation solved in a square or rectangle. Other equations may have different optimum sparsity factors.
FIGURE 5.2

Euclidean error norm against sparsity factor
5.4 THE SPFI SEMI-ITERATIVE METHOD

In the last section it was proposed that one method of solving the model problem (5.6) would be:

i) To find an approximate solution by the SPFI method and then

ii) Find the true solution iteratively.

These two distinctive procedures may be incorporated into different semi-iterative schemes as follows:

a) The stationary first order semi-iterative method,
\[ u_{n+1} = u_n + \alpha \times \text{SPFI}(bAu_n) \]  
(5.12)

b) The stationary second order semi-iterative method,
\[ u_{n+1} = u_n + \alpha \times \text{SPFI}(bAu_n) + \beta (u_n - u_{n-1}) \]  
(5.13)

c) The non-stationary first order semi-iterative method,
\[ u_{n+1} = u_n + \alpha_n \times \text{SPFI}(bAu_n) \]  
(5.14)

d) and finally, the non-stationary second order semi-iterative method
\[ u_{n+1} = u_n + \alpha_n \times \text{SPFI}(bAu_n) + \beta_n (u_n - u_{n-1}) \]  
(5.15)

where \( \alpha_n \) and \( \beta_n \) are acceleration parameters.

The method (5.12) may be written as,
\[ u_{n+1} = u_n + \alpha P(bAu_n) \]  
(5.16)

where \( P \) is the SPFI. The preconditioning method introduced in §4.0 is equivalent to (5.16) but with \( P \) as \((I-\omega L)^{-1}(I-\omega U)^{-1}\). Hence the methods (5.12)-(5.15) are essentially preconditioning techniques with the SPFI being the preconditioning multiplier \( P \).

The question immediately arises as to whether the formulae derived in Chapter 3 for the optimum \( \alpha_n \) and \( \beta_n \) still hold true for these preconditioned SPFI semi-iterative methods. Experimental results, shown in the next section, suggest that they do.
5.5 EXPERIMENTAL ANALYSIS

The problem which is solved throughout this section is again the model system (5.6) with the matrix A of the form (5.11).

Consider first the stationary methods (5.12) and (5.13). The results shown in Table 5.3 were obtained by finding the optimum $\alpha$ (i.e. the least number of iterations) by the method (5.12), and then using this value in method (5.13) to obtain an optimum $\beta$.

**MATRIX A OF ORDER 49**

<table>
<thead>
<tr>
<th>SPARSITY FACTOR</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>LEAST NUMBER OF ITERATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>49</td>
<td>1.0</td>
<td>0.0</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>1.0</td>
<td>0.0</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>0.0</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>1.1</td>
<td>0.0</td>
<td>16</td>
</tr>
</tbody>
</table>

**TABLE 5.3**

As the matrix A is of order 49, it is expected that the process will converge after one iteration when the sparsity factor has value 49 as the method is then direct. Examination of the above table shows this not to be true. This is due to round off error.

Figure 5.3 shows a plot of the time taken for the method (5.13) to converge to the true solution when different sparsity factors are used. A minimum at SPARSITY FACTOR = 3 bears out the conclusions made in §5.3 that an optimum sparsity factor lies in the range 3 to 5. Other results which have been obtained for higher and lower orders of the model matrix confirm that the optimum sparsity factor is in this range.
MATRICE A - ORDER 49

FIGURE 5.3

Timings for method (6.73) against sparsity factor

SPARSITY FACTOR

TIME [SECS]
In Chapter 3 it was shown that the optimum $a_n$ and $b_n$ could be found by substituting the largest and smallest eigenvalues of the coefficient matrix $A$ into the formulae which were derived in that chapter. For those formulae to be valid in the SPFI case the largest and smallest eigenvalues of the matrix SPFI*$A$ would have to be used. As this matrix is unsymmetric it may have complex eigenvalues [Forsythe (1967)], and so the modulus of the eigenvalues,

$$\sqrt{\text{(real part)}^2 + \text{(complex part)}^2}$$

is used instead. Some of these are shown in the table below.

<table>
<thead>
<tr>
<th>SPARSITY FACTOR</th>
<th>MODULUS OF LARGEST EIGENVALUE</th>
<th>MODULUS OF SMALLEST EIGENVALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1.0013</td>
<td>0.9981</td>
</tr>
<tr>
<td>4</td>
<td>1.0014</td>
<td>0.9982</td>
</tr>
<tr>
<td>5</td>
<td>1.0015</td>
<td>0.9982</td>
</tr>
<tr>
<td>10</td>
<td>1.0000</td>
<td>0.9999</td>
</tr>
</tbody>
</table>

**TABLE 5.4**

Modulus of eigenvalues of SPFI*$A$ where $A$ has order 49.

Putting these values into the formula (3.20) an optimum value of $a=1.0$ is obtained in all the cases considered, which corresponds to those in Table 5.4 which were obtained experimentally. Continuing further, and substituting these eigenvalues into the formulae (3.26) for the optimum $a$ and $b$ of the second order stationary Richardson's method it is seen that the values of $a$ and $b$ so obtained again agree with those found experimentally.

Further analysis of this model problem for different orders (up to 150) of the coefficient matrix $A$ show that whatever the order, the optimum $a$ and $b$ have approximate values one and zero respectively in the range $3 < S < N$ where $N$ is the order of the matrix. Hence, even though the formulae (3.20) and (3.26) give the correct optimum $a$ and $b$, they are redundant in this case as the $a$ and $b$ remain the same whatever the order or sparsity factor of SPFI*$A$.

Turning now to the non-stationary semi-iterative methods (5.14) and
(5.15) and performing a similar analysis as for the stationary methods, it is seen that the first order case gives the same results i.e. the same number of iterations as those in Table 5.3. However, for the second order non-stationary case, the procurement of the $\alpha_i$ and $\beta_i$ for use in this method by the formulae (3.28) is very difficult. This difficulty is caused by the fact that the largest and smallest eigenvalues are very close together thus causing $\psi_0$ in (3.28) to be very large. Hence, as successive $\alpha_i$ and $\beta_i$ are calculated, larger and larger numbers have to be dealt with, eventually leading to an overflow situation. The second order non-stationary semi-iterative method is therefore not suitable for solving the model problem.

5.6 COMPUTATIONAL COMPARISON

Having shown experimentally that the SPFI semi-iterative methods converge to the true solution of the model system it is now necessary to compare the iterative work, i.e. (no. of iterations) * (no. of arithmetic ops./iteration) of these schemes with the work involved in using established techniques. The method from each class of schemes which is used in this comparison process is:

1) The stationary second order SPFI semi-iterative method (5.13)
and 2) Richardson's second order method (3.20).

As in §4.3,

No. of arithmetic operations = no. of adds. + 2 * no. of mults.

where it is assumed that,

multiplication $\equiv$ 2 additions in time.

The number of arithmetic operations per iteration required by the two schemes to solve an $N \times N$ model system over a unit square is:

For SPFI semi-iterative method with sparsity factor $r$,

$$\frac{37N}{2} + 3\sqrt{N(N - \frac{9}{2})} + 3Nr - 3r(\frac{r-1}{2})$$  (5.17)

The derivation of (5.17) is shown on page 58a.
and for Richardson's second order method,

\[ 21N - 12\sqrt{N} \]  

(5.18)

From (5.17) and (5.18) a measure of computational work can be obtained for the two techniques being investigated thus enabling a comparison to be carried out.

<table>
<thead>
<tr>
<th>NO. OF OPERATIONS</th>
<th>OPERATIONS PER ITERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPFI METHOD SF=3</td>
<td>RICHARDSON'S ITERATIVE METHOD</td>
</tr>
<tr>
<td>NO. OF ITERATIONS</td>
<td>PER ITERATION</td>
</tr>
<tr>
<td>49</td>
<td>10</td>
</tr>
<tr>
<td>100</td>
<td>14</td>
</tr>
<tr>
<td>144</td>
<td>17</td>
</tr>
</tbody>
</table>

**TABLE 5.5**

Solving Poisson's equation in a unit square

Using an optimum SF of 3 in the SPFI method, it can be seen from the above table that in solving the model problem, some computational gain is achieved in using the SPFI method for the lower orders. However, for the higher orders Richardson's method requires less work. It must be stressed, however, that the actual work involved in obtaining the SPFI has not been included in making the above comparisons and if this is taken into consideration then Richardson's method is by far the better of the two.

In similar tests conducted on rectangular regions the same conclusions as above were also drawn.

It is envisaged that the SPFI method will be beneficial to those problems for which the factorisation of the matrix has already been completed or is easily obtained, e.g. in solving a system with many right hand sides.
Each iteration of the stationary second order semi-iterative method,

\[ u^{(n+1)} = u^{(n)} + \alpha \cdot \text{SPFI}(b-Au^{(n)}) + \beta(u^{(n)} + u^{(n-1)}) , \]  

(5.18)

requires,

\[
\begin{align*}
5N - 4rN & \quad \text{multiplications} \\
5N - 4rN & \quad \text{additions}
\end{align*}
\]

to evaluate \( b-Au^{(n)} \)

\[
\begin{align*}
N\sqrt{N} - \sqrt{N}/2 - N/2 + Nr - \frac{r(r-1)}{2} & \quad \text{mults.} \\
N\sqrt{N} - \sqrt{N}/2 - 3N/2 + Nr - \frac{r(r-1)}{2} & \quad \text{adds.}
\end{align*}
\]

to evaluate \( \text{SPFI}(b-Au^{(n)}) \)

\[
\begin{align*}
2N & \quad \text{multiplications} \\
2N & \quad \text{additions}
\end{align*}
\]

to evaluate \( u^{(n+1)} \)

This gives a total of

\[ 13N/2 + \sqrt{N}(N-9/2) + Nr - \frac{r(r-1)}{2} \] multiplications

and\[ 11N/2 + \sqrt{N}(N-9/2) + Nr - \frac{r(r-1)}{2} \] additions

for each iteration.

Let,

\[ \text{no. of arithmetic operations} = \text{no. of adds.} + 2 \cdot \text{no. of mults.} \]

then the number of operations for each iteration of (5.18) would be

\[ 37N/2 + 3\sqrt{N}(N-9/2) + 3Nr - \frac{3r(r-1)}{2} \]
CHAPTER 6

CONCLUSIONS
6.0 INTRODUCTION

In the research undertaken, there were two distinct areas of interest,

(i) The use of preconditioning in solving a Toeplitz system of Simultaneous linear equations,
and

(ii) The sparse product form of the inverse and its use in iterative methods.

As these were researched independently, the conclusions are thus split into two separate sections covering the above two headings.

6.1 THE USE OF PRECONDITIONING IN SOLVING A TOEPLITZ SYSTEM OF SIMULTANEOUS
LINEAR EQUATIONS

In the past, preconditioning has mainly been used on symmetric coefficient matrices with the effect that the P-condition number of an associated matrix is minimised thus optimising the rate of convergence of the iterative methods being used. However, it has been known for a number of years, that this preconditioning procedure could be used successfully on unsymmetric matrices though there has always been some doubt as to whether an optimum acceleration parameter could be predicted from the eigenvalues of the unsymmetric iteration matrix. The reason for this uncertainty is that an unsymmetric matrix usually has complex eigenvalues.

This thesis has gone some way to clearing up the above contention by looking at a simple unsymmetric coefficient matrix (the Toeplitz matrix of the form (4.7)). As this matrix has complex eigenvalues then the same is true for the preconditioned iteration matrix.

In tackling the problem of predicting the optimum acceleration parameter from the eigenvalues of the iteration matrix Q, two alternatives were considered:
(i) The modulus of the eigenvalues of $Q$ were used and the optimum acceleration parameter $\bar{w}$ defined as that value of $\omega$ for which the $P$-condition number of $Q$ is minimised.

(ii) The eigenvalues of $Q^T Q$, which are all real, were considered and the optimum acceleration $\bar{w}$ defined as that value of $\omega$ for which the $K$-condition number of $Q^T Q$ is minimised.

From experimental analysis, it was shown that the first alternative proved the more accurate of the two although for Toeplitz matrices of order greater than 100 even this method cannot be relied upon to give an accurate $\bar{w}$.

It was also shown that the preconditioned iterative method for the Toeplitz matrix requires less operations than the accelerated Jacobi method which tends to suggest that this conclusion is true for all the established iterative techniques introduced in Chapter 3.

Finally, it should be stated that major work still needs to be carried out in the area of preconditioned unsymmetric matrices. It may be that the restriction on the size of the matrix, so that an optimum acceleration parameter can be found from the eigenvalues of the iteration matrix, is peculiar only to Toeplitz matrices. The authenticity of this statement can only be verified by further research.

6.2 THE SPARSE PRODUCT FORM OF THE INVERSE

The sparse product form of the inverse (SPFI) appears to be an effective means of retaining an approximate inverse of a matrix (in product form) and at the same time minimising the storage requirement. The number of elements of the SPFI which needs to be retained is governed by the sparsity factor and for the model problem considered in Chapter 5 this had an optimum value of between 3 and 5.
Although the SPFI is more difficult to program than the PFI, this disadvantage is outweighed by the storage saving.

The SPFI has to be incorporated into some iterative method for the solution of a system of simultaneous linear equations to be found. This leads to a new set of semi-iterative methods (5.12)-(5.16) which conform to the formulae introduced in Chapter 3 for finding the optimum acceleration parameters of established extrapolated iterative methods.

For the model problem investigated in Chapter 5, (i.e. Poisson's equation solved over a unit square with Dirichlet boundary conditions) it appears that some computational gain is achieved by using the SPFI semi-iterative method as opposed to the established extrapolated iterative techniques only when low orders of the coefficient matrix of the model system are considered. However, it is envisaged that the reduction in the number of iterations effected by use of the SPFI semi-iterative method will be increased when more difficult problems leading to ill-conditioned sparse matrices of high order are tackled.

The results obtained in Chapter 5 look promising, although extensive experiments on a wider range of problems need to be carried out before any further conclusions can be made.

6.3 Suggestions for Further Work

In making conclusions about the work carried out in this thesis, it is clear that a substantial amount of research still needs to be done in the two areas that were covered.

Several unsymmetric matrices of various orders need to be investigated before any definite conclusions can be made about preconditioning of unsymmetric matrices. This is a long and arduous task and would require
more than the year which was afforded the author for the completion of this thesis.

It is also suggested that in the area of the sparse product form of the inverse, several ill-conditioned sparse matrices should be investigated, hence providing further evidence on which conclusions could be made as to the "usefulness" of the SPFI approach.
REFERENCES


Numerical methods for partial differential equations,
Thomas Nelson and Sons Ltd.

Benson, A. (1969)

The numerical solution of partial differential equations by finite
difference methods,
Ph.D. Thesis, University of Sheffield.

Bodewig, E. (1956)

Matrix Calculus
Interscience, New York.

Evans, D.J. (1973)

Comparison of the convergence rates of iterative methods for solving
linear equations with preconditioning,

Forsythe, G.E., Moler, C.B. (1967)

Computer Solution of linear algebraic systems,
Prentice-Hall Inc., Englewood Cliffs, N.J.

Geiringer, H. (1949)

On the solution of systems of linear equations by certain iterative
methods,
Golden, J.T. (1965)

*Programming and Computing,*
Prentice Hall.

Jennings, A. (1977)

*Matrix computation for Engineers and Scientists*
Wiley.


*Mathematical methods for digital computers,*

Smith, G.D. (1968)

*Numerical solution of partial differential equations: Finite difference methods,*

Steifel, E.L. (1958)

*Kernal polynomials in linear algebra and their numerical application,*

Westlake, J.R. (1968)

*Numerical matrix inversion and solution of linear equations,*
John Wiley & Sons, Inc.

Young, D.M. (1954a)

*On Richardson's method for solving linear systems with positive definite matrices,*
Young, D.M., (1954b)

Iterative methods for solving partial differential equations of elliptic type,

Young, D.M. (1971)

Iterative solution of large linear systems,

Axelsson (1974)

On preconditioning and convergence acceleration in sparse matrix problems,
CERN 74-10 Geneva.

Dubois, P.F.; Greenbaum, A.; Rodrique, G.H. (1979)

Approximating the inverse of a matrix for use in iterative algorithms on vector processors,
Computing 22, 257-268.

Evans, D.J. (1980)

On the solution of certain Toeplitz tridiagonal linear systems,

Walsh, J.E. (1962)

Numerical solution of ordinary and partial differential equations,
Pergamon Press, Editor, L. Fox.
APPENDIX 1

FORTRAN PROGRAMS ILLUSTRATING THE PRECONDITIONING OF A TOEPLITZ MATRIX
THIS PROGRAM PERFORMS THE PRECONDITIONED SIMULTANEOUS ITERATIVE METHOD WITH ALPHA=1 ON THE TOEPLITZ MATRIX OF THE FORM (4.7) THUS ENABLING AN OPTIMUM OMEGA TO BE FOUND.

```fortran
real lelt
integer r,c
dimension x(80),y(80),z(80),b(80)

INPUT SECTION

write(6,10)
10 format('INPUT order, lower and upper elements of matrix')
read(5,*)n,lelt,uelt
write(6,20)
20 format('INPUT start w and step size')
read(5,*)w1,stp
write(6,30)
30 format('INPUT the number of steps required')
read(5,*)n2
eps=0.000005
alpha=1.0

GET ACCELERATION

do 998 k=1,n2
w=w1
w=w+(k-1)*stp
zl=lelt*w
zu=uelt*w

SET UP RHS

b(1)=1.0+uelt
do 1 i=2,n-1
b(i)=1.0+uelt-lelt+(zl*b(i-1))
1 continue
b(n)=1.0-lelt+(zl*b(n-1))

MAKE INITIAL GUESS AT Z SAY 0.2 AND HENCE FORM Y

do 2 i=1,n-1
y(i)=0.2+(zu*0.2)
2 continue
y(n)=0.2

WORK OUT B*Y AND START ITERATIONS

r=0
200 c=0
```
FIRST FIND Z FROM \((I+WU)Z=Y\)

\[
z(n)=y(n) \\
i=n-1 \\
250 \ z(i)=y(i)-(zu*z(i+1)) \\
i=i-1 \\
\text{if}(i.<.0) \ \text{goto} \ 250
\]

NEXT FIND X FROM \(A*Z=X\)

\[
x(1)=z(1)+uelt*z(2) \\
do \ 3 \ i=2,n-1 \\
x(i)=z(i)+(uelt*z(i+1))-(lelt*z(i-1)) \\
\text{continue} \\
x(n)=z(n)-lelt*z(n-1)
\]

LAST FIND Z FROM \((I-WL)Z=X\)

\[
z(1)=x(1) \\
do \ 4 \ i=2,n \\
z(i)=x(i)+(zl*z(i-1)) \\
\text{continue}
\]

WORK OUT NEW Y’S AND COMPARE WITH OLD

\[
do \ 5 \ i=1,n \\
x(i)=y(i)+alpha*(b(i)-z(i)) \\
z1=abs(y(i)-x(i)) \\
z2=1+abs(y(i)) \\
z3=z1/z2 \\
\text{if}(z3.1t.eps) \ c=c+1 \\
y(i)=x(i) \\
\text{continue} \\
r=r+1 \\
\text{if}(r.<.500) \ \text{goto} \ 600 \\
\text{if}(c.ne.n) \ \text{goto} \ 200
\]

CONVERT BACK TO ORIGINAL FORM

\[
x(n)=y(n) \\
i=n-1 \\
300 \ x(i)=y(i)-(zu*x(i+1)) \\
i=i-1 \\
\text{if}(i.<.0) \ \text{goto} \ 300
\]

OUTPUT RESULTS

\[
\text{write}(6,400)r,w,alpha \\
400 \ \text{format}('the \ no. \ of \ iterations=',i4,'the \ val \ of \ w=',f7.3,'the \ val \ of \ alpha=',f7.3) \\
\text{write}(6,500)(x(i),i=1,n) \\
500 \ \text{format}(20f5.2) \\
998 \ \text{continue} \\
\text{goto} \ 800 \\
600 \ \text{write}(6,700) \\
700 \ \text{format}('TOO MANY ITERATIONS') \\
800 \ \text{stop}
\]
THIS PROGRAM PERFORMS THE PRECONDITIONED SIMULTANEOUS ITERATIVE METHOD ON THE TOEPLITZ MATRIX OF THE FORM (4.7) USING AN OPTIMUM OMEGA FOUND IN THE PREVIOUS PROGRAM ON PAGE 66. AN OPTIMUM VALUE OF ALPHA CAN THUS BE FOUND

```
c
real uelt
integer r,c
dimension x(80),y(80),z(80),b(80)

c
INPUT SECTION

c write(6,10)
10 format('INPUT order, lower and upper elements of the matrix')
read(5,*)n,uelt,uelt
write(6,15)
15 format('INPUT the optimum value of omega')
read(5,*)w
write(6,20)
20 format('INPUT start alpha and step size')
read(5,*)alpha,step
write(6,30)
30 format('INPUT the number of steps required')
read(5,*)n2
eps=0.000005
z1=uelt*w
zu=uelt*w

c
GET ACCELERATION

c do 998 k=1,n2
alpha=alpha1
alpha=alpha+°k-1°*step

c
SET UP RHS

c b(1)=1.0+uelt
do 1 i=2,n-1
b(i)=1.0+uelt-uelt+(z1*b(i-1))
1 continue
b(n)=1.0-uelt+(z1*b(n-1))

c MAKE INITIAL GUESS AT Z SAY 0.2 AND HENCE FORM Y

c do 2 i=1,n-1
y(i)=0.2+(zu*0.2)
2 continue
y(n)=0.2
```
WORK OUT B*Y AND START ITERATIONS

r = 0

FIRST FIND Z FROM (I+WU)Z=Y

z(n) = y(n)
i = n-1
z(i) = y(i) - (zu*z(i+1))
i = i-1
if (i.gt.0) goto 250

NEXT FIND X FROM A*Z=X

x(1) = z(1) + uelt*z(2)
do 3 i = 2, n-1
x(i) = z(i) + (uelt*z(i+1)) - (1elt*z(i-1))
3 continue
x(n) = z(n) - 1elt*z(n-1)

LAST FIND Z FROM (I-WL)Z=X

z(1) = x(1)
do 4 i = 2, n
z(i) = x(i) + (zl*z(i-1))
4 continue

WORK OUT NEW Y'S AND COMPARE WITH OLD

do 5 i = 1, n
x(i) = y(i) + alpha*(b(i) - z(i))
z1 = abs(y(i) - x(i))
z2 = 1 + abs(y(i))
z3 = z1/z2
if (z3 .lt. eps) c = c + 1
y(i) = x(i)
5 continue
r = r + 1
if (r .gt. 500) goto 600
if (c .ne. n) goto 200

CONVERT BACK TO ORIGINAL FORM

x(n) = y(n)
i = n-1
x(i) = y(i) - (zu*x(i+1))
i = i-1
if (i .gt. 0) goto 300
```fortran
C OUTPUT RESULTS

write(6,400)r,w,alpha
400 format(\"the no. of iterations=\",i4,/\n\"the val of w=\",f7.3/,\n\"the val of alpha=\",f7.3/)
write(6,500)(x(i),i=1,n)
500 format(20f5.2)
998 continue
goto 800
600 write(6,700)
700 format(\"too many iterations\")
800 stop
end
```
Appendix 2

FORTRAN PROGRAMS ILLUSTRATING THE USE OF THE SPFI APPROACH IN ITERATIVE METHODS
THIS PROGRAM PERFORMS A STATIONARY FIRST ORDER SEMI-ITERATIVE
METHOD ON THE MODEL MATRIX (5.11) USING THE SPFI APPROACH.
FROM THIS AN OPTIMUM VALUE OF ALPHA CAN BE FOUND

implicit double precision (a-h,o-p,s-z)
dimension a(150),b(150),c(150),x(150),y(150),v(150),xm(150,150),
i u(150),s(150),d(150)
integer q,q1,r,rc,c1
common/bl1/a,b,c,x,y,v,xm,n,q,r

INPUT SECTION

write(6,10)
format(’what is the order of the matrix’)  
read(5,20)n
20 format(i4)
write(6,30)
format(’what is the bandwidth’)  
read(5,20)  q
write(6,40)
format(’what is the sparsity factor’)  
read(5,20)r
write(6,44)
format(’what is the starting value of alpha’)  
read(5,*)alpha1
write(6,45)
format(’INPUT the number of steps required’)  
read(5,*)n1

SET UP MATRIX A

q1=q-1
do 50 i=1,n
b(i)=4.0
d(i)=4.0
k=i/q1
k=k*q1
if(k.ne.i)then
  c(i)=-1.0
  d(i)=d(i)-1.0
end if
k=(i-1)/q1
k=(k*q1)+1
if(k.ne.i)then
  a(i)=-1.0
  d(i)=d(i)-1.0
end if
if(i.le.(n-q+1))then
  x(i)=-1.0
  d(i)=d(i)-1.0
  y(i)=-1.0
end if
if(i.gt.(q-1)) d(i)=d(i)-1.0
50 continue
c FIND SPFI

call spfi

do 992 i1=1,n
write(6,991)(xm(i1,j1),j1=1,n)
991 format(20f10.4)
992 continue

c MAKE INITIAL GUESS

c
lar=9999999

do 130 k=1,n1
alpha=alpha1
alpha=alpha+(k-1)*0.1
do 53 i=1,n
u(i)=0.2
53 continue
rc=0

c START ITERATIONS

c
55 continue
cl=0

c FIND S\textsuperscript{N}

c
s(1)=d(1)-(b(1)*u(1)+c(1)*u(2)+x(1)*u(q))
do 60 i=2,n-1
s(i)=a(i)*u(i-1)+b(i)*u(i)+c(i)*u(i+1)
if(i.ge.(n-q+1)) s(i)=s(i)+x(i)*u(i+q-1)
if(i.ge.q) s(i)=s(i)+y(i-q+1)*u(i-q+1)
else
s(i)=d(i)-s(i)
60 continue
s(n)=d(n)-(a(n)*u(n-1)+b(n)*u(n)+y(n-q+1)*u(n-q+1))
c

c FIND SPFI*S\textsuperscript{N}

c
li=n-q

do 90 j=1,n
if(j.le.li) then
k1=q+j-1
else
k1=n
end if

do 80 i=1,k1
if(i.le.(j-r)) goto 80
70 if(i.ne.j) then
s(i)=s(i)+xm(i,j)*s(j)
else
p=s(i)*xm(i,j)
end if
80 continue
s(j)=p
90 continue
c FIND U^{N+1}
c
do 100 i=1,n
   t=u(i)+s(i)*alpha
   if(abs(t-u(i)).lt.(0.00005)) cl=cl+1
   u(i)=t
100 continue
rc=rc+1
if(cl.ne.n) goto 55
write(6,120)rc,alpha
if(rc.lt.lar)then
   lar=rc
   alpha2=alpha
end if
130 continue
write(6,110)u(i),i=1,n
110 format(20f10.4)
write(6,120)lar,alpha2
120 format('the no. of iterations = ',i4,'alpha=',f5.2)
stop
end

subroutine spfi
implicit double precision (a-h,o-p,s-z)
dimension a(150),b(150),c(150),x(150),y(150),v(150),xm(150,150)
integer q,r
common/b1/a,b,c,x,y,v,xm,n,q,r
c FIND FIRST COL OF SPFII
xm(1,1)=1/b(1)
xm(2,1)=-a(2)/b(1)
xm(q,1)=-y(1)/b(1)
c PUT NEXT COL OF MATRIX IN VECTOR V
do 500 i=2,n
do 50 j=1,n
   v(j)=0.0
50 continue
   if(i.ge.q) v(i-q+1)=x(i-q+1)
   if(i.le.(n-q+1)) v(q+i-1)=y(i)
v(i-1)=c(i-1)
v(i)=b(i)
   if(i.ne.n) v(i+1)=a(i+1)
c MULTIPLY BY PREVIOUS I-1 COLS OF SPFII
c
11=n-q
do 200 j=1,i-1
if(j.le.11) then
   k1=q+j-1
   else
      k1=n
   end if
end if
do 100 k=1,k1
if(k.le.(j-r)) goto 100
if(j.ne.k) then
   v(k)=v(k)+xm(k,j)*v(j)
   else
      p=v(k)*xm(k,j)
   end if
100 continue
v(j)=p
200 continue

WORK OUT NEXT COL OF SPFI

p=1/v(i)
if(i.le.11) then
   k1=q+i-1
   else
      k1=n
end if

300 continue
xm(j,i)=v(j)*p
300 continue
xm(i,i)=p
500 continue
return
end
THIS PROGRAM PERFORMS A STATIONARY SECOND ORDER SEMI-ITERATIVE
METHOD ON THE MODEL MATRIX (5.11) USING THE SPFI APPROACH.
AN OPTIMUM ALPHA FOUND FROM THE PROGRAM ON PAGE 71 IS USED
THUS ENABLING AN OPTIMUM BETA TO BE FOUND.

NOTE: THE SUBROUTINE SPFI IS THE SAME AS IN THE LAST PROGRAM
AND HAS THUS BEEN OMITTED

implicit double precision (a-h,o-p,s-z)
dimension a(197),b(197),c(197),x(197),y(197),v(197),xm(197,197),
1 u(197),s(197),f(197),d(197)
integer q,q1,r,rc,c1
common/bl1/a,b,c,x,y,v,xm,n,q,r

INPUT SECTION
write(6,10)
10 format(’what is the order of the matrix’)
read(5,20)n
20 format(i4)
write(6,30)
30 format(’what is the bandwidth’)
read(5,20)q
write(6,40)
40 format(’what is the sparsity factor’)
read(5,20)r
write(6,42)
42 format(’what is the optimum value of alpha’) 
read(5,*)alpha
write(6,44)
44 format(’what is the starting value of beta’) 
read(5,*)betal
write(6,45)
45 format(’INPUT number of steps required’) 
read(5,*)nl

SET UP MATRIX A
q1=q-1
do 50 i=1,n
b(i)=4.0
d(i)=4.0
k=1/q1
k=k*q1
if(k.ne.i)then
   c(i)=-1.0
d(i)=d(i)-1.0
end if
50 k=(i-1)/q1
k=(k*q1)+1
if(k.ne.i)then
   a(i)=-1.0
d(i)=d(i)-1.0
end if
if(i.le.(n-q+1)) then
  x(i)=-1.0
  d(i)=d(i)-1.0
  y(i)=-1.0
end if
if(i.gt.(q-1)) d(i)=d(i)-1.0
50 continue

! FIND SPFI 
! MAKE INITIAL GUESS

lar=9999999
do 130 k=1,n1
  beta=beta1
  beta=beta+(k-1)*0.1
  do 53 i=1,n
    u(i)=0.2
    rc=0
    53 continue
  continue
  cl=0
  s(1)=d(1)-(b(1)*u(1)+c(1)*u(2)+x(1)*u(q))
  do 60 i=2,n-1
    s(i)=a(i)*u(i-1)+b(i)*u(i)+c(i)*u(i+1)
    if(i .le. (n-q+1) ) s(i)=s(i)+x(i)*u(i+q-1)
    if(i .ge. q) s(i)=s(i)+y(i-q+1)*u(i-q+1)
    s(i)=d(i)-s(i)
    60 continue
    s(n)=d(n)-(a(n)*u(n-1)+b(n)*u(n)+y(n-q+1)*u(n-q+1))
! FIND SPFI*S^N

l1=n-q
do 90 j=1,n
  if(j .le. l1) then
    k1=q+j-1
    else
      k1=n
  end if
  90 continue

do 80 i=1,k1
if(i.le.(j-r)) goto 80
70 if(i.ne.j) then
   s(i)=s(i)+xm(i,j)*s(j)
   else
     p=s(i)*xm(i,j)
   end if
continue
s(j)=p
90 continue
c

c FIND U^{N+1}
c
do 100 i=1,n
   t=u(i)+s(i)*alpha+(u(i)-f(i))*beta
   if(abs(t-u(i)).lt.(0.00005)) cl=cl+1
   f(i)=u(i)
   u(i)=t
100 continue
rc=rc+1
if(cl.ne.n) goto 55
write(6,120)rc,alpha,beta
if(rc.lt.lar) then
   lar=rc
   beta2=beta
end if
130 continue
write(6,110)(u(i),i=1,n)
110 format(20f10.4)
write(6,120)lar,alpha,beta2
120 format('the no. of iterations = ',i4,'alpha=',f5.2,'beta=',f5.2)
stop
end
This program performs a non-stationary first order semi-iterative method on the model matrix (5.11) using the SPFI method. The value of \( m \), the sorting factor (see page 22) must be even.

Note: Here again the subroutine SPFI has been omitted. It is the same as that on page 73.

Implicit double precision (a-h,o-p,s-z)
Dimension a(50),b(50),c(50),x(50),y(50),v(50),xm(50,50),d(50),
1 u(50),s(50),f(50),alpha(50)
Integer q,q1,r,rc,cl
Common/b11/a,b,c,x,y,v,xm,n,q,r/b12/alpha,m,lar,sml

Input section

Write(6,10)
10 Format(‘what is the order of the matrix’) Read(5,20)n
20 Format(i4) Write(6,30)
30 Format(‘what is the bandwidth’) Read(5,20)q
40 Format(‘what is the sparsity factor’) Read(5,20)r
45 Format(‘what is the value of m’) Read(5,*)m
47 Format(‘what is the largest and smallest e’vals of matrix’) Read(5,*)lar,sml

Find alpha’s

\( m1=(m+1)/2 \)
\( nadd=1 \)
\( nsub=0 \)
\( nswi=1 \)
Call sort

Set up matrix A

\( q1=q-1 \)
Do 50 i=1,n
b(i)=4.0
d(i)=4.0
k=i/q1
k=k*q1
If(k.ne.i)then
  c(i)=-1.0
  d(i)=d(i)-1.0
end if
k=(i-1)/q1
k=(k*q1)+1
if(k.ne.i)then
   a(i)=-1.0
   d(i)=d(i)-1.0
end if
if(i.le.(n-q+1))then
   x(i)=-1.0
   d(i)=d(i)-1.0
   y(i)=-1.0
end if
if(i.gt.(q-1)) d(i)=d(i)-1.0
continue

FIND SPFI

call spfi

MAKE INITIAL GUESS

lar=9999999
do 53 i=1,n
   u(i)=0.2
53 continue
rc=0

START ITERATIONS

continue
c1=0

FIND S^N

s(1)=d(1)-(b(1)*u(1)+c(1)*u(2)+x(1)*u(q))
do 60 i=2,n-1
s(i)=a(i)*u(i-1)+b(i)*u(i)+c(i)*u(i+1)
if(i.le.(n-q+1)) s(i)=s(i)+x(i)*u(i+q-1)
if(i.ge.q) s(i)=s(i)+y(i-q+1)*u(i-q+1)
s(i)=d(i)-s(i)
do 60 continue
s(n)=d(n)-(a(n)*u(n-1)+b(n)*u(n)+y(n-q+1)*u(n-q+1))

FIND SPFI*S^N

do 90 j=1,n
do 80 i=1,n
if(i.le.(j-r)) goto 80
if(i.ne.j) then
   s(i)=s(i)+xm(i,j)*s(j)
else
   p=s(i)*xm(i,j)
end if
80 continue
s(j)=p
90 continue

c OBTAIN CORRECT ORDER FOR ALPHA'S

c if(nswi.eq.0) then
   np=ml+nadd
   nadd=nadd+1
   nswi=1
else
   np=ml-nsub
   nsub=nsub+1
   nswi=0
end if
if(np.eq.1) then
   nswi=1
   nadd=1
   nsub=0
end if
do 100 i=1,n

c FIND U^N+1

t=u(i)+s(i)*alpha(np)
if(abs(t-u(i)).lt.(0.00005)) cl=cl+1
f(i)=u(i)
u(i)=t
100 continue
rc=rc+1
if(cl.ne.n) goto 55
write(6,110)(u(i),i=1,n)
110 format(20f10.4)
write(6,120)rc
120 format('the no. of iterations = ',i4)
stop
end

subroutine sort

c THIS SUBROUTINE CALCULATES ALPHA'S VIA THE FORMULA ON PAGE 22

c AND Sorts THEM IN DESCENDING ORDER

c implicit double precision (a-h,o-p,s-z)
dimension alpha(50)
common/b12/alpha,m,lar,sml
pi=22.0/7.0
r=lar+sml
s=lar-sml
do 50 i=1,m
z=((2*i-1)*pi)/2*m
alpha(i)=2/(r-s*cos(z))
50 continue
do 200 i=1,m-1
big=alpha(i)
loc=i
do 100 j=i+1,m
if(alpha(j).le.big) goto 100
big=alpha(j)
loc=j
100 continue
temp=alpha(i)
alpha(i)=alpha(loc)
alpha(loc)=temp
200 continue
write(6,*)(alpha(i),i=1,m)
return
end