The numerical solution of elliptic partial differential equations by finite difference methods

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

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/10433

Publisher: © Moira Joan Biggins

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

For the full text of this licence, please go to: http://creativecommons.org/licenses/by-nc-nd/2.5/
THE NUMERICAL SOLUTION OF ELLIPTIC
PARTIAL DIFFERENTIAL EQUATIONS
BY FINITE-DIFFERENCE METHODS

by

MOIRA JOAN BIGGINS, B.Sc.

A Doctoral Thesis
Submitted in partial fulfilment of the requirements
for the award of Doctor of Philosophy
of the Loughborough University of Technology
May, 1980.

Supervisor: PROFESSOR D.J. EVANS, Ph.D.,D.Sc.,
Department of Computer Studies

© by Moira Joan Biggins, 1980.
DECLARATION

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

M.J. BIGGINS.
ACKNOWLEDGEMENTS

I would like to thank Professor D.J. Evans, whose guidance and encouragement steered me through this project.

I would also like to thank Miss Judith Briers for her fast and accurate typing of the thesis.

My gratitude extends to the staff of Loughborough University Computer Centre for their help and efficiency in running all the programs, and to the Science Research Council for financial support.
## INDEX

1. **INTRODUCTION**
   - 1.1 Fundamentals of Partial Differential Equations  
   - 1.2 Finite-difference Approximations to Derivatives
     - 1.2.1 Finite-difference Approximations in Cartesian Co-ordinates
     - 1.2.2 Finite-difference Approximations in Equilateral-triangular Co-ordinates
     - 1.2.3 Finite-difference Approximations in Skew Co-ordinates
   - 1.3 Fundamental Definitions and Theorems
     - 1.3.1 Diagonal Dominance and Irreducibility
     - 1.3.2 Positive Definite Matrices
     - 1.3.3 Vector and Matrix Norms
     - 1.3.4 Convergence of Sequences of Matrices
     - 1.3.5 Property (A) and Consistent Ordering
     - 1.3.6 The Power Method

2. **FUNDAMENTALS OF ITERATIVE METHODS FOR SPARSE SYSTEMS OF LINEAR EQUATIONS**
   - 2.1 Introduction
   - 2.2 Basic Iterative Methods
     - 2.2.1 The Jacobi Method
     - 2.2.2 The Point Gauss-Seidel and SOR Methods
     - 2.2.3 The Block Gauss-Seidel and SOR Methods
     - 2.2.4 Experimental Results for SOR on the Model Problem
     - 2.2.5 Methods of Simultaneous Iteration
     - 2.2.6 Preconditioning
     - 2.2.7 Preconditioned Methods of Simultaneous Iteration
2.3 Computational Efficiency of Iterative Methods

2.3.1 Relative Execution Times of Floating-point instructions

2.3.2 Comparison of Efficiency of Iterative Methods

2.4 Experimental Results for Preconditioning with Mikhlin and Smolitsky's Finite-difference Approximation

3. SUCCESSIVE OVER-RELAXATION TECHNIQUES APPLIED TO PROBLEMS IN NON-CARTESIAN GEOMETRIES

3.1 Introduction

3.2 Poisson's Equation in an Equilateral Triangle

3.2.1 Modified Periodic Tridiagonal Algorithms

3.2.2 Experimental Results for Point SOR and SPOR

3.2.3 Variations of and Extensions to the Basic SPOR Method

3.2.4 A Note on Computer Implementation of Problems in a Triangular Region

3.3 Poisson's Equation in a Parallelogram

3.3.1 A Modified Seven-diagonal Algorithm

3.3.2 Experimental Results for Point and Block SOR

4. TECHNIQUES FOR THE SOLUTION OF MILDLY NON-LINEAR EQUATIONS

4.1 Introduction

4.2 Inner/Outer Iteration Strategies

4.3 Experimental Results for Point and Block SOR Methods

4.4 The Linearisation Technique for Mildly Non-linear Equations

4.5 Adaptive Techniques for the Determination of the Optimum Over-relaxation Factor of the SOR Method

4.5.1 Kulrsrud's Technique

4.5.2 Carre's Technique

4.5.3 Reid's Technique

4.5.4 Conclusions

4.6 Preconditioned Methods of Simultaneous Iteration
5. SUCCESSIVE OVER-RELAXATION AND PRECONDITIONED ITERATIVE TECHNIQUES APPLIED TO A CERTAIN FORMULATION OF LAPLACE'S EQUATION IN THE UNIT SQUARE

5.1 Introduction 215
5.2 The Four-point Block 217
5.3 Experimental Results for the Four-point Block SOR Method 226
5.4 Experimental Results for Preconditioned Methods of Simultaneous Iteration Applied to the Explicit Four-point Block Matrix 230

6. AN APPROXIMATE METHOD OF PRECONDITIONED SIMULTANEOUS DISPLACEMENT

6.1 Introduction 239
6.2 The Explicit Line Block Matrix 240
6.3 Preconditioning and Approximate Preconditioning 242
6.4 A Preconditioned Approximate Method 248

SUMMARY 256
REFERENCES 258
APPENDIX: SELECTED PROGRAM LISTINGS 261
CHAPTER 1

INTRODUCTION
1.1 FUNDAMENTALS OF PARTIAL DIFFERENTIAL EQUATIONS

Many problems in engineering and physics involve the rates of change of unknown quantities, such as pressure or temperature, with respect to two or more independent variables, typically time, length, and angle. The mathematical formulation of these problems leads either to a partial differential equation (p.d.e.) or a set of such equations.

The exact solution of a p.d.e. in a region \( R \) with boundary \( B \) is a function which satisfies the equation at every point in \( R \) and also matches the specified conditions (called the boundary conditions) on \( B \). As an example, suppose we wish to solve a second-order p.d.e. in the bounded, connected, plane region \( R \) with boundary \( B \), and that the solution on \( B \) must match a function \( t(x,y) \) which is defined on \( B \). The problem is to find a function \( \phi(x,y) \) which is continuous in \( R+B \), twice differentiable in \( R \) and satisfies in \( R \) the general second-order p.d.e.

\[
\frac{\partial^2 \phi}{\partial x^2} + 2b \frac{\partial^2 \phi}{\partial x \partial y} + c \frac{\partial^2 \phi}{\partial y^2} + d \frac{\partial \phi}{\partial x} + e \frac{\partial \phi}{\partial y} + f \phi = g \tag{1.1a}
\]

and on \( B \) the condition

\[
\phi(x,y) = t(x,y) \tag{1.1b}
\]

(Alternatively, the boundary conditions can be specified on \( B \) in terms of the normal derivative \( \frac{\partial \phi}{\partial n} \) or by a linear combination of \( \phi \) and \( \frac{\partial \phi}{\partial n} \).

In (1.1a), if the coefficients are constants or functions of the independent variables \( x \) and \( y \), the equation is linear. If they include the dependent variable \( \phi \) or its derivatives, the equation is non-linear. A non-linear equation in which \( a, b, \) and \( c \) are functions of \( x, y, \phi \), and its first-order derivatives is called quasi-linear. If \( a, b \) and \( c \) are functions of \( x \) and \( y \), the equation is described as mildly non-linear. A linear or quasi-linear equation in which

\[
d = \frac{\partial a}{\partial x} + \frac{\partial b}{\partial y}, \quad e = \frac{\partial b}{\partial x} + \frac{\partial c}{\partial y}
\]

is called self-adjoint.
(Special forms of (1.1) occur very frequently, since they describe conservation principles in physics).

Second-order partial differential equations are classified according to the relationships between the coefficients $a$, $b$ and $c$. The equation is said to be

- **elliptic** if $b^2 - ac < 0$,
- **parabolic** if $b^2 - ac = 0$,
- **hyperbolic** if $b^2 - ac > 0$.

\[ (1.2) \]

Some typical examples are:

**Elliptic**

- Poisson's equation,
- Laplace's equation,

\[
\begin{align*}
\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} &= \{ g(x,y) \} \\
\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} &= 0
\end{align*}
\]

\[
(\frac{\partial^2 }{\partial x^2} + \frac{\partial^2 }{\partial y^2} \text{ is called the Laplacian operator and written } \nabla^2),
\]

**Parabolic**

- The heat conduction equation \( \frac{\partial \phi}{\partial t} = \frac{\partial^2 \phi}{\partial x^2} \).

**Hyperbolic**

- The wave equation \( \frac{\partial^2 \phi}{\partial t^2} = \frac{\partial^2 \phi}{\partial x^2} \).

The elliptic equations are usually associated with steady-state problems. For example, Laplace's equation describes the velocity potential for the steady flow of incompressible, non-viscous fluid, and is the mathematical expression of the physical law that the rate at which such fluid enters a given region is equal to the rate at which it leaves it.

Such problems are called **boundary value problems**, since it is usual to have the function $\phi$ (say) specified either explicitly or implicitly (by means of the normal derivative) at all points on the boundary of the region.
For arbitrarily-shaped regions and general boundary conditions, it is usually impossible to determine an exact solution to a given p.d.e. Even if an exact solution can be found, it may require an unacceptably large amount of computation to evaluate. As an example, consider the Poisson equation $V^2 \phi = -2$, which describes the torsion on a long solid elastic prism whose rectangular cross-section is defined by $x = \pm a$, $y = \pm b$. Its solution (obtained by the technique of separation of variables) is

$$\phi = b^2 - y^2 - 32b^2 \pi^{-3} \sum_{n=0}^{\infty} \left\{ \frac{(-1)^n}{(2n+1)^3} \sech \frac{(2n+1)\pi a}{2b} \cosh \frac{(2n+1)\pi x}{2b} \cos \frac{(2n+1)\pi y}{2b} \right\}.$$ 

In an attempt to obtain solutions to elliptic partial differential equations with a reasonable amount of computation, approximate methods have been developed. These are of two types:

(i) Analytical approximate methods where, for example, a truncated series may be found for the solution. Although these methods can provide useful information about the character of the solution for critical values of the dependent variables, they tend to be difficult to apply and are not always valid over the whole region under consideration. They will not be discussed further in this thesis.

(ii) Numerical approximate methods, such as finite-difference, finite-element and boundary-element methods, which are the subject of this thesis, involve making discrete approximations to the p.d.e.s and are especially suited to solving problems in regular regions. Finite-element methods involve making discrete approximations to the connectivity of the elements of the region itself, and are particularly useful for dealing with irregular regions. The boundary element method is a boundary integral equation technique which involves discretizing the boundary of the region and is well suited to problems where results are required near and on the boundary of the region.
1.2 FINITE-DIFFERENCE APPROXIMATIONS TO DERIVATIVES

Let us consider, without loss of generality, that the problem is to solve an elliptic p.d.e. with dependent variable \( \phi \) and independent variables \( x \) and \( y \), the region of the problem being a connected region \( R \) in the \( x-y \) plane. \( x \) and \( y \) denote some suitable co-ordinate system; the systems used in this thesis are Cartesian, equilateral-triangular and skew. Let \( \overline{R} = R + B \) denote the closure of the region \( R \) with boundary \( B \). We overlay \( \overline{R} \) with a system of meshes formed by two sets of lines, one set parallel to \( Ox \) and the other parallel to \( Oy \). The intersections of these lines are referred to here as the (unknown) mesh points. Other names used in the literature are grid, lattice, pivotal and nodal points. The p.d.e. is approximated at each of the \( M \) mesh points internal to \( B \) by an algebraic equation involving the values of \( \phi \) at the mesh point itself and certain neighbouring mesh points. This leads to a set of \( M \) equations (linear, if the original equation is linear) in \( M \) unknowns. The solution of this set of equations is then an approximate solution of the p.d.e.

The three sub-sections which follow describe the derivation of the finite-difference approximations used in this thesis. The sub-sections deal respectively with Cartesian, equilateral-triangular and skew geometries.

First, we define the symbol \( O \) to mean \( f(t) = O(g(t)) \) as \( t \to a \) if there exists a number \( k \) such that

\[
\frac{|f(t)|}{|g(t)|} < k
\]

for all \( t \) sufficiently close to \( a \).

1.2.1 Finite-difference approximations in Cartesian co-ordinates

Consider a uniform Cartesian grid of mesh size \( h \), illustrated in Figure 1.1 on the unit square, with the mesh points numbered as shown.
Assuming that \( \phi(x,y) \) is sufficiently differentiable, then by Taylor's theorem we have

\[
\phi(x \pm h, y) = \phi(x, y) \pm h \frac{\partial \phi}{\partial x} + \frac{h^2}{2!} \frac{\partial^2 \phi}{\partial x^2} + h^2 \frac{\partial^3 \phi}{\partial y \partial x^3} + \cdots, \quad (1.3)
\]

\[
\phi(x, y \pm h) = \phi(x, y) \pm h \frac{\partial \phi}{\partial y} + \frac{h^2}{2!} \frac{\partial^2 \phi}{\partial y^2} + h^2 \frac{\partial^3 \phi}{\partial x \partial y^3} + \cdots, \quad (1.4)
\]

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

\[
+ \frac{h^3}{3!} \left( \frac{\partial^3 \phi}{\partial x \partial y^2} + \frac{\partial^3 \phi}{\partial x^3} + \frac{\partial^3 \phi}{\partial x \partial y^3} + \cdots \right)
\]

\[
+ \frac{h^4}{4!} \left( \frac{\partial^4 \phi}{\partial x^4} + \frac{\partial^4 \phi}{\partial x^2 \partial y^2} + \cdots \right) + \cdots \quad (1.5)
\]
and

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

where the points \((x \pm h, y), (x, y \pm h)\) and \((x \pm h, y \pm h)\) are contained in \(\mathbb{R}\) and all derivatives are evaluated at \((x, y)\).

Equations (1.3) and (1.4) respectively give

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

(1.7)

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

(1.8)

(1.9)

(1.10)

(More accurate approximations are

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

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

Combinations of equations (1.3) and (1.4) give approximations to the second-order derivatives

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

(1.11)

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

(1.12)

A combination of equations (1.5) and (1.6) gives an approximation to the second-order mixed derivative

\[ \frac{\partial^2 \phi}{\partial x \partial y} = \frac{\phi(x+h, y-h) - \phi(x-h, y+h) - \phi(x-h, y-h) - \phi(x+h, y-h)}{4h^2} + O(h^2) \]
Adding equations (1.11) and (1.12) gives a finite-difference approximation to $V^2\phi$. If for the general mesh point $(x_i,y_j)=(ih,jh)$ we denote $\phi(x_i,y_j)$ by $\phi_{i,j}$, then Poisson's equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = g(x,y)$$

(1.14)
can be replaced at the point $(x_i,y_j)$ by the finite-difference equation

$$\frac{1}{h^2}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - 4\phi_{i,j}) = g_{i,j} + \frac{h^4}{12}(\frac{\partial^4 \phi}{\partial x^4} + \frac{\partial^4 \phi}{\partial y^4}) + \ldots,$$

(1.15)

which on multiplying through by $-h^2$ becomes

$$4\phi_{i,j} - 4\phi_{i+1,j} - 4\phi_{i-1,j} - 4\phi_{i,j+1} - 4\phi_{i,j-1} = -h^2g_{i,j} + \frac{h^4}{12}(\frac{\partial^4 \phi}{\partial x^4} + \frac{\partial^4 \phi}{\partial y^4}) - \ldots,$$

(1.16)
The terms on the right-hand side of (1.16), excluding $(-h^2g_{i,j})$, are defined as the \textit{local truncation error} of this formula. The $O(h^4)$ term is the \textit{principal part} of this error. If we scan over the mesh points with formula (1.16), omitting the local truncation error, we obtain a set of simultaneous equations whose solution $\{z_{i,j}\}$ is a finite-difference approximation of the exact solution $\{\phi_{i,j}\}$ at the internal mesh points.

As an illustration, consider the unit square with mesh size $\frac{1}{4}$ (nine internal points) and boundary conditions $t(x,y)$. This gives the set of equations

$$\begin{bmatrix}
4 & -1 & -1 \\
-1 & 4 & -1 \\
-1 & 4 & -1 \\
-1 & -1 & 4 \\
-1 & 4 & -1 \\
-1 & -1 & 4 \\
-1 & -1 & 4 \\
-1 & -1 & 4 \\
-1 & -1 & 4
\end{bmatrix} \begin{bmatrix}
z_{11} \\
z_{12} \\
z_{13} \\
z_{21} \\
z_{22} \\
z_{23} \\
z_{31} \\
z_{32} \\
z_{33}
\end{bmatrix} = \begin{bmatrix}
-h^2g_{11} + t_{10} + t_{01} \\
-h^2g_{12} + t_{02} \\
-h^2g_{13} + t_{14} + t_{03} \\
-h^2g_{21} + t_{20} \\
-h^2g_{22} \\
-h^2g_{23} + t_{24} \\
-h^2g_{31} + t_{30} + t_{41} \\
-h^2g_{32} + t_{42} \\
-h^2g_{33} + t_{34} + t_{43}
\end{bmatrix},$$

(1.17)
which can be written
\[ Az = b \]  
(1.17')

In general, if there are \( N^2 \) internal mesh points, then \( z \) and \( b \) are \((N^2 \times 1)\) vectors and \( A \) is a matrix of order \( N^2 \).

From equation (1.16), we see that the smaller the value of \( h \), the better the accuracy of the finite-difference approximation. However, a decrease in \( h \) means an increase in the number of simultaneous equations to be solved. (In a square region, \( h^{-1} = N+1 \); in other regions and geometries the exact relationship between mesh size and number of points may differ).

Mikhlin and Smolitsky (1967), pp.72-73 give a finite-difference approximation to the self-adjoint elliptic p.d.e.

\[
\frac{\partial^2}{\partial x^2}(a\frac{\partial \phi}{\partial x} + e \frac{\partial \phi}{\partial y}) + \frac{\partial}{\partial y}(b \frac{\partial \phi}{\partial x} + b \frac{\partial \phi}{\partial y}) = g(x,y),
\]
(1.18)

where \( a(x,y), b(x,y), \) and \( e(x,y) \) are such that \( a>0, b>0, ab-e^2>0 \). The approximation will be derived here for \( a, b \) and \( e \) constant.

The first derivatives are replaced by the formulae (1.7) and (1.9) (forward-difference approximations). Thus,

\[
\frac{\partial}{\partial x}(a \frac{\partial \phi}{\partial x} + e \frac{\partial \phi}{\partial y}) = \frac{1}{h}(\phi_{i+1,j} - \phi_{i,j}) + \frac{e}{h}(\phi_{i,j+1} - \phi_{i,j})
\]
(1.19)

and

\[
\frac{\partial}{\partial y}(b \frac{\partial \phi}{\partial x} + b \frac{\partial \phi}{\partial y}) = \frac{1}{h}(\phi_{i+1,j} - \phi_{i,j}) + \frac{b}{h}(\phi_{i,j+1} - \phi_{i,j}).
\]
(1.20)

The second differentiation is performed using the operators given in formulae (1.8) and (1.10) (backward-difference approximations) and leads to

\[
\frac{\partial}{\partial x}(a \frac{\partial \phi}{\partial x} + e \frac{\partial \phi}{\partial y}) = \frac{1}{h^2}(\phi_{i+1,j+1} - \phi_{i,j+1} + \phi_{i,j+1} - \phi_{i-1,j+1})
\]

\[
+ \frac{e}{h^2}(\phi_{i,j+1} - \phi_{i-1,j+1} + \phi_{i+1,j} - \phi_{i-1,j})
\]

\[
= \frac{1}{h^2}(a(\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j})
\]

\[
+ e(\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i+1,j})
\]
(1.21)
and

\[
\frac{\partial^2}{\partial y^2}(\frac{\partial \phi}{\partial x} + \phi \frac{\partial \phi}{\partial y}) = \frac{e}{h^2}((\phi_{i+1,j} - \phi_{i,j+1} - \phi_{i,j-1}) - (\phi_{i,j} - \phi_{i-1,j})) \\
+ \frac{b}{h^2}((\phi_{i,j+1} - \phi_{i,j}) - (\phi_{i,j} - \phi_{i-1,j})) \\
- \frac{1}{h^2}((\phi_{i+1,j} - \phi_{i,j+1} - \phi_{i,j-1}) - (\phi_{i,j} - \phi_{i-1,j})) \\
+ b(\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1})].
\]  

(1.22)

The finite-difference representation of equation (1.18) is then

\[
c^{(0)}\phi_{i,j} + c^{(1)}\phi_{i+1,j} + c^{(2)}\phi_{i,j+1} + c^{(3)}\phi_{i,j-1} - c^{(4)}\phi_{i-1,j} = -g_{i,j},
\]

(1.23a)

where

\[
c^{(0)} = \frac{2(a+b+e)}{h^2}, \quad c^{(1)} = \frac{a+e}{h^2}, \quad c^{(2)} = \frac{b+e}{h^2}, \quad c^{(3)} = \frac{e}{h^2}, \quad c^{(4)} = \frac{a+e}{h^2}, \quad c^{(5)} = \frac{b+e}{h^2}, \quad c^{(6)} = \frac{e}{h^2}.
\]

(1.23b)

The left-hand side of equation (1.23a) can be written as,

\[
c^{(0)}\phi_{i,j} + c^{(1)}(\phi_{i+1,j} + \phi_{i,j+1} + \phi_{i,j-1}) + c^{(2)}(\phi_{i,j+1} + \phi_{i,j-1}) - c^{(4)}(\phi_{i-1,j} + \phi_{i,j-1}).
\]

(1.24)

Replacing the \( \phi \) terms by their Taylor series expansions about \( (i,j) \), as given in equations (1.3), (1.4), (1.5) and (1.6), leads to

\[
\frac{2(a+b+e)}{h^2}\phi_{i,j} \\
- \frac{a+e}{h^2} (2\phi_{i,j} + a^2 \frac{\phi}{\partial x} + h^4 \frac{\phi}{\partial x^4} + \ldots) \\
- \frac{b+e}{h^2} (2\phi_{i,j} + a^2 \frac{\phi}{\partial y} + h^4 \frac{\phi}{\partial y^4} + \ldots) \\
+ \frac{e}{h^2} (2\phi_{i,j} + a^2 \frac{\phi}{\partial x} + \frac{2}{a^2} \phi + \frac{2}{a} \phi) \\
+ h^4 (a^2 \frac{\phi}{\partial x^4} - a^2 \frac{\phi}{\partial x^2} \frac{\phi}{\partial y} - a^2 \frac{\phi}{\partial y^2} + \frac{2}{a} \phi + \phi) \\
+ \frac{2}{a^2} \phi + \frac{2}{a} \phi + \frac{2}{a} \frac{\phi}{\partial x} + \frac{2}{a} \frac{\phi}{\partial y} + \phi + \phi + \ldots)
\]
Hence the local truncation error of formula (1.23) is $O(h^2)$.

1.2.2 Finite-Difference Approximations in Equilateral-Triangular Co-ordinates

First we define the relationship between Cartesian and triangular co-ordinates, and express the Laplacian operator in triangular co-ordinates. Let $x$ and $y$ denote the Cartesian axes, and $u, v$ and $w$ the triangular axes. Assume that the $u$-axis coincides with the $x$-axis, and that the $v$- and $w$-axes make angles of $\alpha$ and $\beta$ respectively with it, as shown in Figure 1.2.

The relationship between the two geometries is defined by the equations

$$x = u + v \cos \alpha + w \cos \beta$$

and

$$y = v \sin \alpha + w \sin \beta.$$  

\begin{equation}
= -\left( \frac{\partial^2 \phi}{\partial x^2} + 2\frac{\partial^2 \phi}{\partial x \partial y} + \frac{\partial^2 \phi}{\partial y^2} \right) + O(h^2).
\end{equation}
The partial derivatives of \( x \) and \( y \) with respect to \( u, v \) and \( w \) are therefore,
\[
\frac{\partial x}{\partial u} = 1, \quad \frac{\partial x}{\partial v} = \cos \alpha, \quad \frac{\partial x}{\partial w} = \cos \beta,
\]
\[
\frac{\partial y}{\partial u} = 0, \quad \frac{\partial y}{\partial v} = \sin \alpha, \quad \frac{\partial y}{\partial w} = \sin \beta. \tag{1.27}
\]

A function \( \phi(x,y) \) can be considered a function of \( u,v \) and \( w \) through the intermediate functions \( x \) and \( y \) defined by equations (1.26). Its derivatives can be found by the rule for the differentiation of composite functions. Thus we have
\[
\frac{\partial \phi}{\partial u} = \frac{\partial \phi}{\partial x} \frac{\partial x}{\partial u} + \frac{\partial \phi}{\partial y} \frac{\partial y}{\partial u},
\]
and similarly,
\[
\frac{\partial \phi}{\partial v} = \cos \alpha \frac{\partial \phi}{\partial x} + \sin \alpha \frac{\partial \phi}{\partial y}, \tag{1.28}
\]
Differentiating again, we obtain
\[
\frac{\partial^2 \phi}{\partial u^2} = \left( \frac{\partial \phi}{\partial x} \right)^2 + \left( \frac{\partial \phi}{\partial y} \right)^2
\]
and likewise
\[
\frac{\partial^2 \phi}{\partial v^2} = \cos^2 \alpha \frac{\partial^2 \phi}{\partial x^2} + 2 \sin \alpha \cos \alpha \frac{\partial^2 \phi}{\partial x \partial y} + \sin^2 \alpha \frac{\partial^2 \phi}{\partial y^2}, \tag{1.29}
\]
For equilateral-triangular co-ordinates, \( \alpha=60^0, \beta=120^0 \) and equations (1.29) simplify to
\[
\frac{\partial^2 \phi}{\partial u^2} = \frac{\partial^2 \phi}{\partial x^2},
\]
\[
\frac{\partial^2 \phi}{\partial v^2} = \frac{1}{4} \frac{\partial^2 \phi}{\partial x^2} + \frac{\sqrt{3}}{2} \frac{\partial^2 \phi}{\partial x \partial y} + \frac{3}{4} \frac{\partial^2 \phi}{\partial y^2}, \tag{1.30}
\]
For equilateral-triangular co-ordinates, \( \alpha=60^0, \beta=120^0 \) and equations (1.29) simplify to
\[
\frac{\partial^2 \phi}{\partial u^2} = \frac{\partial^2 \phi}{\partial x^2},
\]
\[
\frac{\partial^2 \phi}{\partial v^2} = \frac{1}{4} \frac{\partial^2 \phi}{\partial x^2} + \frac{\sqrt{3}}{2} \frac{\partial^2 \phi}{\partial x \partial y} + \frac{3}{4} \frac{\partial^2 \phi}{\partial y^2}, \tag{1.30}
\]
These can be combined to give
\[
\frac{2}{3} \frac{\partial \phi}{\partial x} + \frac{2}{3} \frac{\partial \phi}{\partial y} = \frac{2}{3} \left( \frac{\partial \phi}{\partial u} + \frac{\partial \phi}{\partial v} + \frac{\partial \phi}{\partial w} \right)
\]
(1.31)

See Salvadori and Baron (1961), pp. 245-246.

Figure 1.3 shows the unit equilateral triangle overlaid with a triangular mesh. Mesh points are numbered as shown.

![Figure 1.3](image)

**FIGURE 1.3**

Note that while only two subscripts are needed to define a particular mesh point, each point actually has three co-ordinates, and the representation \((u_i, v_j, w_k)\) of a point is not, in general, unique. Figure 1.4 shows the simplest numbering scheme for the mesh point \((u_i, v_j, w_k)\) and its immediate neighbouring mesh points.
If we assume that the function \( \phi \) is sufficiently differentiable, Taylor series expansions for the values of \( \phi \) at the neighbouring mesh points are

\[
\phi_{i \pm 1, j, k} = \phi_{i, j, k} + h \frac{\partial \phi}{\partial u} + \frac{h^2}{2!} \frac{\partial^2 \phi}{\partial u^2} + \frac{h^3}{3!} \frac{\partial^3 \phi}{\partial u^3} + \frac{h^4}{4!} \frac{\partial^4 \phi}{\partial u^4} + \cdots,
\]

\[
\phi_{i, j \pm 1, k} = \phi_{i, j, k} + h \frac{\partial \phi}{\partial v} + \frac{h^2}{2!} \frac{\partial^2 \phi}{\partial v^2} + \frac{h^3}{3!} \frac{\partial^3 \phi}{\partial v^3} + \frac{h^4}{4!} \frac{\partial^4 \phi}{\partial v^4} + \cdots,
\]

and

\[
\phi_{i, j, k \pm 1} = \phi_{i, j, k} + h \frac{\partial \phi}{\partial w} + \frac{h^2}{2!} \frac{\partial^2 \phi}{\partial w^2} + \frac{h^3}{3!} \frac{\partial^3 \phi}{\partial w^3} + \frac{h^4}{4!} \frac{\partial^4 \phi}{\partial w^4} + \cdots.
\]  

(1.32)

Combinations of these yield

\[
\frac{\partial^2 \phi}{\partial u^2} = \frac{2}{h^2} (\phi_{i+1, j, k} - 2\phi_{i, j, k} + \phi_{i-1, j, k}) + O(h^2),
\]
\[
\frac{\partial^2 \phi}{\partial v^2} = \frac{2}{h} (\phi_{i,j+1,k} - 2\phi_{i,j,k} + \phi_{i,j-1,k}) + O(h^2),
\]

and

\[
\frac{\partial^2 \phi}{\partial w^2} = \frac{2}{h} (\phi_{i,j,k+1} - 2\phi_{i,j,k} + \phi_{i,j,k-1}) + O(h^2).
\]

Hence \( \nabla^2 \phi = g(u,v,w) \) can be replaced at the point \((u_i,v_j,w_k)\) by

\[
\frac{2}{3h^2} (\phi_{i+1,j,k} + \phi_{i-1,j,k} + 2\phi_{i,j+1,k} + 2\phi_{i,j-1,k} + 2\phi_{i,j,k+1} + 2\phi_{i,j,k-1}) - 6\phi_{i,j,k} = g_{i,j,k} + O(h^2),
\]

or, multiplying through by \( -\frac{3h^2}{2} \),

\[
6\phi_{i,j,k} - \phi_{i+1,j,k} - \phi_{i-1,j,k} - \phi_{i,j+1,k} - \phi_{i,j-1,k} - \phi_{i,j,k+1} - \phi_{i,j,k-1} = -\frac{3h^2}{2} g_{i,j,k} + O(h^4).
\]

The local truncation error of this formula is \( O(h^4) \).

### 1.2.3 Finite-Difference Approximations in Skew Co-ordinates

First we define the relationship between Cartesian and skew geometries, and express the Laplacian operator in skew co-ordinates. Let \( x \) and \( y \) denote the Cartesian axes, and \( u \) and \( v \) the skew axes. Assume that the \( u \)-axis coincides with the \( x \)-axis, and that the \( v \)-axis makes an angle \( \alpha \) with it, as shown in Figure 1.5.

![Figure 1.5](image)
The relationship between the two geometries is defined by the equations

\[ x = u + v \cos \alpha \]

and

\[ y = v \sin \alpha \quad (1.36) \]

The partial derivatives of \( \phi \) can be found in a similar way to those in the previous section. The second-order derivatives are

\[ \frac{\partial^2 \phi}{\partial u^2} = \frac{\partial^2 \phi}{\partial x^2}, \]

\[ \frac{\partial^2 \phi}{\partial v^2} = \cos^2 \alpha \frac{\partial^2 \phi}{\partial x^2} + 2 \sin \alpha \cos \alpha \frac{\partial^2 \phi}{\partial x \partial y} + \sin^2 \alpha \frac{\partial^2 \phi}{\partial y^2} \]

and

\[ \frac{\partial^2 \phi}{\partial u \partial v} = \cos \alpha \frac{\partial \phi}{\partial x} + \sin \alpha \frac{\partial \phi}{\partial y}. \quad (1.37) \]

These equations can be combined to give,

\[ \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = \frac{1}{\sin \alpha} \left\{ \frac{\partial^2 \phi}{\partial u^2} - 2 \cos \alpha \frac{\partial^2 \phi}{\partial u \partial v} + \frac{\partial^2 \phi}{\partial v^2} \right\}. \quad (1.38) \]


Figure 1.6 shows the unit rhombus with acute angle \( \alpha \), overlaid with a skew mesh whose mesh lengths are different in the two direction. The mesh points are numbered as shown.
Assuming $\phi$ to be sufficiently differentiable, Taylor series expansions for the mesh points neighbouring $(u_i, v_j)$ are:

\[
\phi_{i \pm 1, j} = \phi_{i, j} \pm h \frac{\partial \phi}{\partial u} + h^2 \frac{\partial^2 \phi}{\partial u^2} + \frac{h^3}{3!} \frac{\partial^3 \phi}{\partial u^3} + \frac{h^4}{4!} \frac{\partial^4 \phi}{\partial u^4} + \cdots,
\]

\[
\phi_{i, j \pm 1} = \phi_{i, j} \pm k \frac{\partial \phi}{\partial v} + \frac{k^2}{2!} \frac{\partial^2 \phi}{\partial v^2} + \frac{k^3}{3!} \frac{\partial^3 \phi}{\partial v^3} + \frac{k^4}{4!} \frac{\partial^4 \phi}{\partial v^4} + \cdots,
\]

\[
\phi_{i \pm 1, j \pm 1} = \phi_{i, j} \pm h \frac{\partial \phi}{\partial u} \pm k \frac{\partial \phi}{\partial v} + \frac{1}{2!} (h^2 \frac{\partial^2 \phi}{\partial u^2} \pm 2hk \frac{\partial^2 \phi}{\partial u \partial v} + k^2 \frac{\partial^2 \phi}{\partial v^2})
\]

\[
+ \frac{1}{3!} (h^3 \frac{\partial^3 \phi}{\partial u^3} \pm 3h^2 k \frac{\partial^3 \phi}{\partial u^2 \partial v} + 3hk^2 \frac{\partial^3 \phi}{\partial u \partial v^2} \pm k^3 \frac{\partial^3 \phi}{\partial v^3})
\]

\[
+ \frac{1}{4!} (h^4 \frac{\partial^4 \phi}{\partial u^4} \pm 4h^3 k \frac{\partial^4 \phi}{\partial u^3 \partial v} + 6h^2 k^2 \frac{\partial^4 \phi}{\partial u^2 \partial v^2} \pm 4hk^3 \frac{\partial^4 \phi}{\partial u \partial v^3} \pm k^4 \frac{\partial^4 \phi}{\partial v^4} + \cdots
\]

and

\[
\phi_{i, j \pm 1} = \phi_{i, j} \pm h \frac{\partial \phi}{\partial u} \pm k \frac{\partial \phi}{\partial v} + \frac{1}{2!} (h^2 \frac{\partial^2 \phi}{\partial u^2} \pm 2hk \frac{\partial^2 \phi}{\partial u \partial v} + k^2 \frac{\partial^2 \phi}{\partial v^2})
\]

\[
- \frac{1}{3!} (h^3 \frac{\partial^3 \phi}{\partial u^3} \pm 3h^2 k \frac{\partial^3 \phi}{\partial u^2 \partial v} + 3hk^2 \frac{\partial^3 \phi}{\partial u \partial v^2} \pm k^3 \frac{\partial^3 \phi}{\partial v^3})
\]

\[
+ \frac{1}{4!} (h^4 \frac{\partial^4 \phi}{\partial u^4} \pm 4h^3 k \frac{\partial^4 \phi}{\partial u^3 \partial v} + 6h^2 k^2 \frac{\partial^4 \phi}{\partial u^2 \partial v^2} \pm 4hk^3 \frac{\partial^4 \phi}{\partial u \partial v^3} \pm k^4 \frac{\partial^4 \phi}{\partial v^4} + \cdots
\]

Combinations of these yield

\[
\frac{\partial^2 \phi}{\partial u^2} = \frac{1}{2} (\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}) + O(h^2),
\]

\[
\frac{\partial^2 \phi}{\partial v^2} = \frac{1}{2} (\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}) + O(h^2)
\]

and

\[
\frac{\partial^2 \phi}{\partial u \partial v} = \frac{1}{4hk} (\phi_{i+1,j+1} - \phi_{i+1,j} - \phi_{i,j+1} + \phi_{i,j}) + O(h^2) + O(k^2)
\]

Hence $v^2 \phi = g(u, v)$ can be replaced at the point $(u_i, v_j)$ by...
\[
\frac{1}{h^2 \sin^2 \alpha} \left( \phi_{i+1,j} \phi_{i-1,j} \right) + \frac{h^2}{k^2} \left\{ \phi_{i,j+1} + \phi_{i,j-1} \right\} - 2 \left\{ \frac{h^2}{k^2} \right\} \phi_{i,j}
\]
\[
\frac{1}{2} \frac{h}{k} \cos \alpha \left\{ \phi_{i+1,j+1} - \phi_{i+1,j-1} - \phi_{i-1,j+1} + \phi_{i-1,j-1} \right\} = g_{i,j} + O(h^2).
\]

(1.41)

Multiplying through by \(-h^2 \sin^2 \alpha\), re-arranging and denoting \(\frac{h}{k}\) by \(\lambda\) gives
\[
2(1+\lambda^2) \phi_{i,j} - (\phi_{i+1,j} + \phi_{i-1,j}) - \lambda^2 (\phi_{i,j+1} + \phi_{i,j-1})
\]
\[
+ \frac{1}{2} \lambda \cos \alpha (\phi_{i+1,j+1} - \phi_{i+1,j-1} - \phi_{i-1,j+1} + \phi_{i-1,j-1}) = -h^2 g_{i,j} \sin^2 \alpha + O(h^4).
\]

(1.42)

The local truncation error of this formula is \(O(h^4)\).
1.3 FUNDAMENTAL DEFINITIONS AND THEOREMS

The application of a finite-difference method to the solution of a p.d.e. such as (1.14) leads to a system of simultaneous equations such as (1.17'). This system has a unique solution \( \mathbf{z} = \mathbf{A}^{-1}\mathbf{b} \) provided \( \mathbf{A} \) is non-singular. The matrix \( \mathbf{A} \) is sparse i.e. the majority of its elements are zero. We see from equations (1.16) and (1.17), for example, that a formula involving five mesh points gives a coefficient matrix with at most five non-zero elements per row.

This thesis deals with methods of solving such systems of equations. The methods depend on the properties of the matrix \( \mathbf{A} \), for example, irreducibility, diagonal dominance and positive definiteness. These and other properties are defined in the remainder of this section, generally without proof (although references to proofs are given). We have presupposed a basic knowledge of matrix and linear algebra theory, which is thoroughly covered in such books as Fox (1964).

1.3.1 Diagonal Dominance and Irreducibility

**Definition 1.1**

An \((n \times n)\) matrix \( \mathbf{A} \) is *diagonally dominant* if

\[
|a_{i,i}| \geq \sum_{j=1}^{n} |a_{i,j}|, \text{ for all } 1 \leq i \leq n. \tag{1.43}
\]

It is strictly diagonally dominant if strict inequality in (1.43) is valid for all \( 1 \leq i \leq n \).

**Definition 1.2**

An \((n \times n)\) matrix \( \mathbf{A} \) is *irreducible* if \( n = 1 \) or if \( n > 1 \) and given any two non-empty disjoint subsets \( S \) and \( T \) of \( W \), the set of the first \( n \) positive integers, such that \( S + T = W \), there exists \( i \in S \) and \( j \in T \) such that \( a_{i,j} \neq 0 \).
This definition is given by Young (1971), p.37. Varga (1962), pp.18-19, defines the (1×1) matrix to be irreducible only if the single element is non-zero. Young (ibid), p.37 gives as a theorem (and proves) the following alternative definition.

**Definition 1.3**

A is irreducible if and only if there does not exist a permutation matrix $P$ such that $P^{-1}AP$ has the form

$$P^{-1}AP = \begin{pmatrix} F & 0 \\ G & H \end{pmatrix}$$

where $F$ and $H$ are square matrices and $0$ is the null matrix.

Neither Definition 1.2 nor Definition 1.3 leads to a simple and systematic method of determining whether a given matrix is irreducible. However, such a method does exist and is described below.

**Definition 1.4**

Let $A$ be an $(n×n)$ matrix, and consider any $n$ distinct points in the plane, which we shall call nodes. For every non-zero element $a_{i,j}$ of the matrix, connect the node $P_i$ to the node $P_j$ by means of a path $P_i \rightarrow P_j$, directed from $P_i$ to $P_j$, as in Figure 1.7. (For non-zero diagonal elements $a_{i,i}$ the path goes from $P_i$ to itself forming a loop, as in Figure 1.8). The resulting diagram is called a finite directed graph, $G(A)$ of $A$.

![Figure 1.7](image-url)
As an example, consider the matrix
\[
\begin{bmatrix}
2 & -1 & 0 \\
-1 & 2 & -1 \\
0 & -1 & 2
\end{bmatrix}
\]

Its directed graph is

**Definition 1.5**

A directed graph is *strongly connected* if for any ordered pair of nodes \( P_i \) and \( P_j \), there exists a directed path

\[
\overrightarrow{P_1 k_1 \rightarrow P_i k_1 \rightarrow P_i k_2 \rightarrow \ldots \rightarrow P_i k_{r-1}} \rightarrow P_j
\]

connecting \( P_i \) to \( P_j \).
Clearly, the directed graph in Figure 1.9 is strongly connected.

The next theorem describes the relationship between the irreducibility of a matrix and its directed graph.

Theorem 1.1

A square matrix $A$ is irreducible if and only if its directed graph is strongly connected.

Definition 1.6

An irreducible matrix which is also diagonally dominant with strict inequality holding for at least one $i$ in (1.43) is said to be irreducibly diagonally dominant.

Theorem 1.2

An irreducibly diagonally dominant matrix is non-singular. If all diagonal elements are positive real numbers, then the real parts of its eigenvalues are strictly positive. The proof of this is indicated by Varga (1962), p.23.

1.3.2 Positive Definite Matrices

Definition 1.7

If a matrix $A$ is Hermitian (equal to its conjugate transpose), and $(x, Ax) > 0$ for all $x \neq 0$, then $A$ is positive definite. (N.B. if $x, y$ are n×1 vectors, then

$$(x, y) = \sum_{i=1}^{n} \bar{x}_i y_i$$

where $\bar{x}_i$ is the complex conjugate of $x_i$).

The following theorem is sometimes used as a definition of positive definiteness.

Theorem 1.3

A real matrix is positive definite if and only if it is symmetric and all its eigenvalues are strictly positive.
Theorem 1.4

A real positive definite matrix $A$ has a unique real positive definite square root $B$ such that $B^2 = A$. $B$ is written $A^{\frac{1}{2}}$. This theorem is proved in Young (1971) p.23.

Theorem 1.5

A real symmetric matrix $A$ is positive definite if and only if it can be written in the form $Q^TQ = A$ where $Q$ is a non-singular matrix.

Proof

(i) If $A = Q^TQ$, $(\det Q \neq 0)$, then for any vector $x \neq 0$,
\[
(x, Ax) = x^TAx = x^TQ^TQx = (Qx)^TQx > 0
\]
Hence $A$ is positive definite by Definition 1.7.

(ii) If $A$ is real and positive definite, it has a real, positive definite (and therefore symmetric) square root $A^{\frac{1}{2}}$. $Q = A^{\frac{1}{2}}$ gives the required solution.

Theorem 1.6

An irreducibly diagonally dominant matrix which is also symmetric and has positive real diagonal elements is positive definite. This follows immediately from Theorems 1.2 and 1.3 since the eigenvalues of a symmetric matrix are real.

1.3.3 Vector and Matrix Norms

Definition 1.8

Let $A$ be an $(n \times n)$ matrix with eigenvalues $\lambda_i$, $1 \leq i \leq n$. Then
\[
\rho(A) = \max_i |\lambda_i|
\]
is the spectral radius of $A$.

Definition 1.9

The following quantities are defined as the $1, 2$ and $\infty$ norms of an $(n \times 1)$ vector $x$:
\[ \|x\|_1 = |x_1| + |x_2| + \ldots + |x_n|, \quad (1.44) \]
\[ \|x\|_2 = (|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2)^{\frac{1}{2}}, \quad (1.45) \]
\[ \|x\|_\infty = \max_i |x_i|. \quad (1.46) \]

**Definition 1.10**

Let \( A \) be an \((n \times n)\) matrix. A matrix norm \( \|A\| \) is *compatible* with a vector norm \( \|x\| \) if
\[ \|Ax\| \leq \|A\| \|x\|. \quad (1.47) \]

We seek to derive matrix norms which are compatible with the vector norms. From (1.47), it seems logical to express
\[ \|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}, \quad (1.48) \]
which is equivalent to
\[ \|A\| = \max_{\|x\|=1} \|Ax\|. \quad (1.49) \]

**Definition 1.11**

A matrix norm constructed by means of (1.49) is said to be *subordinate* to the corresponding vector norm.

The matrix norms subordinate to the vector norms given in Definition 1.9 are respectively,
\[ \|A\|_1 = \max_{i=1}^n \sum_j |a_{i,j}|, \quad (1.50) \]
\[ \|A\|_2 = \| \sigma(A^H A) \|^{\frac{1}{2}}, \quad (1.51) \]
(where \( A^H \) is the conjugate transpose of \( A \)),
\[ \|A\|_\infty = \max_{i=1}^n \sum_j |a_{i,j}|. \quad (1.52) \]
From Definition 1.10, it is easy to show that

$$||A|| \geq \rho(A)$$  \hspace{1cm} (1.53)

for an arbitrary \((n \times n)\) matrix \(A\) and arbitrary norm. See Varga (1962), p.10 for the proof.

### 1.3.4 Convergence of Sequences of Matrices

**Definition 1.12**

Let \(A\) be a square matrix. Then \(A\) is **convergent** to the null matrix 0 if the sequence of matrices \(A, A^2, A^3, \ldots\) converges to the null matrix 0, and is **divergent** otherwise.

**Theorem 1.7**

If \(A\) is an arbitrary square matrix, then \(A\) is convergent if and only if \(\rho(A) < 1\).

**Proof**

If

$$J_i = \begin{bmatrix} \lambda_i & 1 \\ \lambda_1 & \ddots & 1 \\ & \ddots & \ddots & 1 \\ & & \lambda_i \\ & & & \lambda_i \end{bmatrix}$$  \hspace{1cm} (1.54)

is a Jordan sub-matrix of \(A\), then there exists a non-singular matrix \(H\) such that \(HAH^{-1}\) is a block tridiagonal matrix, defined by

$$HAH^{-1} = \text{diag}(J_i)$$,

where \(\lambda_i (1 \leq i \leq r)\) are the distinct eigenvalues of \(A\). If each Jordan sub-matrix is raised to the power \(m\), then the result tends to the null matrix as \(m \to \infty\) if and only if \(|\lambda_i| < 1\) \((1 \leq i \leq r)\), i.e., \(\rho(A) < 1\). (This proof is given in greater detail in Varga (1962), pp.13-15).
1.3.5 Property (A) and Consistent Ordering

**Definition 1.13**

A matrix $A$ of order $n$ has property (A) if there exist two disjoint subsets $S$ and $T$ of the first $n$ positive integers $W$ such that $S+T=W$ and if $a_{i,j} \neq 0$ then either $i=j$ or $i \in S$, $j \in T$ or $i \in T$, $j \in S$.

An immediate consequence of this definition is that a matrix $A$ which has property (A) can, by suitable permutation of its rows and columns, be written as

$$
\begin{bmatrix}
D_1 & F \\
G & D_2
\end{bmatrix}
$$

(1.55)

where $D_1$ and $D_2$ are square diagonal matrices.

**Definition 1.14**

A matrix $A$ of order $n$, is consistently ordered if, starting from its ordering, the rows and columns can be permuted in such a way that if $a_{i,j} \neq 0$, the ordering relation between the $i^{th}$ and $j^{th}$ rows is unchanged and the resulting matrix has the form

$$
\begin{bmatrix}
D_1 & F_1 \\
G_2 & D_2 & F_2 \\
G_3 & D_3 & F_3 \\
& & \ddots & \ddots \\
& & & & 0
\end{bmatrix}
$$

(1.56)

where $D_i$ is $(r_i, x_i)$, $1 \leq i \leq p$;

$F_i$ is $(r_i, x_{i+1})$, $1 \leq i \leq p-1$;

$G_i$ is $(r_{i+1}, x_i)$, $2 \leq i \leq p$;

and $\sum_{i=1}^{p} r_i = n$.

This definition is based on the one given by Forsythe and Wasow (1960), p. 244.
An alternative definition of consistent ordering can be made using an ordering vector. Let \( \mathbf{a} = (a_1, a_2, \ldots, a_N) \) be an ordering vector of the matrix \( A \), where the \( a_i \), \( 1 \leq i \leq N \), are integers and \( a_i \) refers to the \( i \)th equation. Then the matrix is consistently ordered if for every \( a_i, a_j \neq 0 \), either \( i > j \) and \( a_i = a_j + 1 \), or \( i < j \) and \( a_i = a_j - 1 \).

1.3.6 The Power Method

The power method is a well-known iterative process for computing the spectral radius of a matrix. Let us consider an \((n \times n)\) matrix \( A \) with independent eigenvectors \( \mathbf{v}_i \), \( i = 1, 2, \ldots, n \) and corresponding eigenvalues \( \lambda_i \), \( i = 1, 2, \ldots, n \). Any arbitrary vector \( \mathbf{x} \) can then be expressed in the form,

\[
\mathbf{x} = \sum_{i=1}^{n} a_i \mathbf{v}_i.
\]

If we define the iterative process

\[
\mathbf{x}^{(k)} = A \mathbf{x}^{(k-1)}, \quad \mathbf{x}^{(0)} \text{ arbitrary}
\]

then since \( \mathbf{x}^{(0)} \) can be expressed in the form (1.57), we have

\[
\mathbf{x}^{(k)} = A \mathbf{x}^{(k-1)} = A^2 \mathbf{x}^{(k-2)} = \ldots = A^k \mathbf{x}^{(0)} = \sum_{i=1}^{n} a_i \lambda_i^k \mathbf{v}_i.
\]

Suppose that \( \lambda_1 \) is the real eigenvalue of largest modulus, and \( |\lambda_1| \neq |\lambda_i| \), \( i = 2, 3, \ldots, n \). Then \( \mathbf{x}^{(k)} \) can be written as

\[
\mathbf{x}^{(k)} = \lambda_1^k \left( a_1 \mathbf{v}_1 + \sum_{i=2}^{n} a_i \left( \frac{\lambda_i}{\lambda_1} \right)^k \mathbf{v}_i \right),
\]

which for sufficiently large \( k \) becomes

\[
\mathbf{x}^{(k)} = \lambda_1^k \left( a_1 \mathbf{v}_1 + \mathbf{e}^{(k)} \right),
\]

where \( \mathbf{e}^{(k)} \) is a vector with very small components. (We assume \( a_1 \neq 0 \).)

When \( \mathbf{e}^{(k)} \) is negligible to the required precision, \( \mathbf{x}^{(k)} \) is the un-normalised eigenvector of \( A \) corresponding to \( \lambda_1 \), and the ratios of
corresponding components of $x^{(k)}$ and $x^{(k-1)}$ have the constant value $\lambda_1$.

In order to keep the elements of $x^{(k)}$ within reasonable bounds during computation to prevent overflow, it is usual to normalise the vector at each iteration by dividing all its elements by the element of largest modulus. The sequence of normalising factors then converges to $\lambda_1$.

We shall frequently need to find the spectral radius of a matrix which has two eigenvalues of largest modulus which are of opposite sign. Suppose these are $\lambda_1$ and $-\lambda_1$, so that after $k$ iterations we have

$$x^{(k)} = \lambda_1^k (a_1 v_1 + a_2 v_2 + \varepsilon^{(k)})$$

and after $k+1$ iterations we have

$$x^{(k+1)} = \lambda_1^{k+1} (a_1 v_1 - a_2 v_2 + \varepsilon^{(k+1)})$$

Even when the $\varepsilon$ terms in the above two equations become negligibly small we cannot obtain an estimate of $\lambda_1$ from the vectors $x^{(k)}$ and $x^{(k+1)}$, because they contain different combinations of the eigenvectors $v_1$ and $v_2$. However, the vector $x^{(k+2)}$ contains the same combination of $v_1$ and $v_2$ as $x^{(k)}$ does, i.e.,

$$x^{(k+2)} = \lambda_1^{k+2} (a_1 v_1 + a_2 v_2 + \varepsilon^{(k+2)})$$

Hence if we normalise after each iteration as described above, the product of the normalising factors at the $(k+1)^{th}$ and $(k+2)^{th}$ iterations is then an approximation to $\lambda_1^2$. 
CHAPTER 2

FUNDAMENTALS OF ITERATIVE METHODS

FOR SPARSE SYSTEMS OF LINEAR EQUATIONS
2.1 INTRODUCTION

This chapter deals with well-known methods of solving large systems of simultaneous equations with sparse coefficient matrices. These methods are applied to novel problems in this chapter and the following two chapters; they are used as the basis of new methods in Chapters 5 and 6. Some experimental results are given for comparison with the new computational techniques. A more detailed approach than usual to comparing the computational efficiency of iterative methods is discussed. Experimental results are given for Mikhlin and Smolitsky's finite-difference approximation (Chapter 1).
2.2 BASIC ITERATIVE METHODS

Consider a region $R$ with boundary $B$ overlaid with a system of meshes as described in Chapter 1. The application of an appropriate finite-difference formula (such as (1.16) or (1.23) in Cartesian geometry, (1.35) in equilateral-triangular geometry or (1.42) in skew geometry) to each mesh point in turn gives a system of simultaneous equations with a sparse coefficient matrix. We take Laplace's equation in Cartesian geometry as a model problem for the purposes of illustration. However, the discussion is applicable to the other problems listed above, unless otherwise stated. We assume that the boundary conditions are specified explicitly, i.e. $\phi(x,y) = t(x,y)$ on $B$ (Dirichlet boundary conditions).

If at any point the boundary does not coincide with a mesh line, finite-difference approximations to derivatives at nearby mesh points involve a mesh length which is smaller than the grid size. This is illustrated for our model problem in Figure 2.1. The point $(i+1,j)$ lies outside the region of solution: hence it is not defined by the problem and cannot be used in a finite-difference approximation to the p.d.e. being solved. Such a point is called a virtual point. The point $(P) = ((i+\theta)h, jh)$ ($0<\theta<1$) lies on the boundary and so $\phi_P$ is known.
(P) is called an *irregular mesh point*. By combining the Taylor series expansions of \( \phi_{i-1,j} \) and \( \phi_p \) about \( \phi_{i,j} \) it can be shown that

\[
\left\{ \frac{\partial \phi}{\partial x} \right\}_{i,j} = \frac{2}{h^2} \left\{ \frac{1}{(1+\theta)} \phi_p + \frac{1}{1+\theta} \phi_{i-1,j} - \frac{1}{\theta} \phi_{i,j} \right\} + O(h) \quad (2.1)
\]

In the example, the mesh length in the \( y \)-direction is regular, so \( \nabla^2 \phi = g(x,y) \) can be replaced at the point \((i,j)\) by

\[
(2 + \frac{2}{\theta}) \phi_{i,j} - \phi_{i,j+1} - \phi_{i,j-1} - \frac{2}{\theta(1+\theta)} \phi_p - \frac{2}{1+\theta} \phi_{i-1,j} = -h^2 g_{i,j} + O(h^3) \quad (2.2)
\]

Throughout the thesis, we assume for convenience that all mesh points are regular.

We now consider a fixed labelling of the mesh points of \( R \). Scanning over the mesh points with any of the finite-difference formulae listed in Section 2.1 in the order specified by this labelling, yields, as described in Chapter 1, a system of equations represented in matrix notation as

\[
A \mathbf{z} = \mathbf{b} \quad (2.3)
\]

Here \( A \) is a real \((N \times N)\) matrix (where \( N \) is the number of unknown mesh points). Its diagonal entries are the coefficients of \( \phi_{i,j} \) in the formula. Its off-diagonal elements consist of the coefficients of the other \( \phi \) terms in the formula which do not correspond to boundary points, or zeros. \( \mathbf{z} \) and \( \mathbf{b} \) are \((N \times 1)\) vectors, \( \mathbf{z} \) consists of the unknown approximate solutions \( z_{i,j} \) and \( \mathbf{b} \) consists of the known boundary values minus some multiple of the right-hand side \( g \).

For equations (1.16) and (1.35) the matrix \( A \) can be shown to have the following properties:

\[
a_{i,i} > 0 \quad \forall i \neq N \quad \text{and} \quad A \text{ is irreducibly diagonally dominant} \quad (2.4)
\]
The latter condition can be proved from Theorem 1.1 and Definition 1.6. It also follows from Theorem 1.2 that A is non-singular, and hence the matrix equation (2.3) has a unique solution. If all the mesh points in \( R+B \) are regular, A is symmetric, and all its eigenvalues are real. Therefore from Theorem 1.6 we know that A is positive definite. (If \( R+B \) contains irregular mesh points, A is not, in general, symmetric).

The matrix derived from the finite-difference representation (1.23) of the p.d.e. (1.18) is not diagonally dominant. However, the p.d.e. can be shown to give rise to a matrix which is positive definite (see Mikhlin and Smolitsky (1967), p.72). Hence the matrix equation (2.3) has a unique solution in this case.

Likewise, the matrix derived from the finite-difference representation (1.42) of the p.d.e. (1.38) is not diagonally dominant, but equation (1.38) has the same form as equation (1.18), which we have already stated gives rise to a positive definite matrix. Therefore the matrix equation (2.3) has a unique solution in this case also.

If we order the internal mesh points in a suitable way, the matrix A will have a block tridiagonal structure.

Consider the model problem in the unit square, with \( m^2 \) internal points ordered column-wise (sometimes called the natural ordering) as in Figure 2.2.

```
(0,0)  1  5  9  13
  2  6 10 14
  3  7 11 15
  4  8 12 16
(0,1)
```

FIGURE 2.2
The matrix then has the structure indicated by (2.5a) and (2.5b).

\[
A = \begin{bmatrix}
B_1 & C_1 & 0 \\
A_2 & C_{n-1} \\
0 & A_n & B_n
\end{bmatrix}
\]  

where \( n=m \),

\[
B_r = \begin{bmatrix}
4 & -1 \\
-1 & 4 & -1 & 0 \\
0 & -1 & 4 & -1
\end{bmatrix}, \quad 1 \leq r \leq m
\]

and

\[
A_{r+1} = C_r = \begin{bmatrix}
-1 \\
-1 & 0 \\
0 & -1 & -1
\end{bmatrix}, \quad 1 \leq r \leq m-1.
\]  

(2.5b)

Alternatively, points may be ordered on pairs of columns, as shown in Figure 2.3.
The matrix derived from this ordering has the block structure shown in (2.5a), where \( n = \frac{m}{2} \),

\[
B_r = \begin{bmatrix}
4 & -1 & -1 \\
-1 & 4 & 0 & -1 \\
-1 & 0 & 4 & -1 & -1 \\
-1 & 4 & 0 & -1 \\
0 & -1 & -1 & 4 & 0 & -1 \\
-1 & 0 & 4 & -1 & -1 \\
-1 & -1 & 4
\end{bmatrix}, \quad 1 \leq r \leq \frac{m}{2}
\]

and

\[
A_{r+1} = C_r = \begin{bmatrix}
0 \\
-1 \\
0 & 0 \\
0 & 0 \\
-1
\end{bmatrix}, \quad 1 \leq r \leq \frac{m}{2}-1. \tag{2.5c}
\]

Orderings of the mesh points exist such that the square sub-matrices \( B_r \) are not of uniform size and hence the \( A_r \) and \( C_r \) sub-matrices are not square. Consider the model problem as before, but with a spiral ordering of points such that each peripheral consists of consecutively numbered points, as shown in Figure 2.4. This ordering was proposed in Benson (1969).

The coefficient matrix has the block structure of (2.5a), with

\[
n = \begin{cases} 
\frac{m}{2}, & m \text{ even} \\
\frac{m+1}{2}, & m \text{ odd}.
\end{cases}
\]
In this case, $B_1$ is a square matrix of order $4(m-1)$, $B_2$ is a square matrix of order $4(m-3)$, etc. and $A_2 = C_1^T$ is a rectangular matrix $(4(m-3) \times 4(m-1))$, etc. The structures of these sub-matrices, illustrated for $B_r$ of order 8 and $A_r = C_{r-1}^T$ $(8 \times 16)$, are:
We now consider the task of computing the solution of a system such as that represented by the matrix equation (2.3). As illustrated in Chapter I, to obtain a high degree of accuracy in the finite-difference equations, it is necessary to use a small mesh size; this leads to the problem of solving a large system of simultaneous equations.

Direct methods of solution such as Gaussian elimination or Cholesky factorization are not suitable, since they do not take into account the sparsity of the coefficient matrix (although for matrices of very simple structure, e.g. tridiagonal, very efficient simple forms of direct methods do exist). Iterative methods, on the other hand, make use of the sparsity of the matrix; the zero elements are never involved in the computation.
In the use of iterative methods, one assumes an initial approximation to the solution and then successively modifies the approximation according to some prescribed rule. The approximation can, in general, be made arbitrarily close to the solution of the difference equations. The choice of the initial approximation is usually arbitrary; very occasionally it may be significant. (An example of the latter is given in Chapter 5).

We consider two classes of iterative methods, which we shall call simultaneous and successive methods. In simultaneous methods, all elements of the approximate solution are modified at the same time, i.e. the \((n+1)^{th}\) iterate is a function of the \(n^{th}\) and earlier iterates only. Hence the ordering of the mesh points is arbitrary. In successive methods the elements of the approximate solution are modified one after the other, using the latest available values of the iterate, so that some elements of the \((n+1)^{th}\) iterate are functions of the \((n+1)^{th}\) and earlier iterates. In this case the ordering of the mesh points is significant. Successive methods are themselves of two types, point iterative methods in which each component of the iterate is modified by an explicit calculation, and block iterative methods in which blocks of equations are modified successively, the blocks themselves being solved simultaneously.

2.2.1 The Jacobi Method

The simplest matrix iterative method is the Jacobi method, which is of simultaneous type. We express the coefficient matrix \(A\) of (2.3) as

\[
A = D - L - U ,
\]

(2.6)

where \(D\) is a positive diagonal matrix whose elements are the diagonal elements of \(A\), and \(-L\) and \(-U\) are the strictly lower triangular and strictly upper triangular components of \(A\) respectively. Equation (2.3) then becomes

\[
(D - L - U)z = b .
\]

(2.7)
Since each diagonal element of $D$ is strictly positive, $D^{-1}$ exists. If we let

$$B = D^{-1}(L+U), \; c = D^{-1}b,$$

then (2.7) can be written

$$z = Bz + c. \quad (2.7')$$

The Jacobi method is defined by

$$z^{(n+1)} = Bz^{(n)} + c, \quad n>0, \quad (2.8)$$

where $z^{(n)}$ denotes the $n$th approximation to the solution.

Convergence of iterative methods

The Jacobi method is one of a large class of iterative methods which can be written in the form

$$z^{(n+1)} = Mz^{(n)} + d. \quad (2.9)$$

$M$ is called the iteration matrix of the method and $d$ is a vector of constants.

Definition 2.1

An iterative process $z^{(n+1)} = Mz^{(n)} + d$ is consistent with a matrix equation $Az = b$ if the set of solutions of the matrix equation is a subset of the set of solutions of the iterative process. It can be shown (see Young (1971), p.65) that if the consistency condition applies, then for some $n$,

$$z^{(n+1)} = Mz^{(n)} + k = Mz + k = z.$$

That is, once the solution is obtained, the iterative process makes no further modification of successive iterates.

Definition 2.2

An iterative method converges if, for any given $b$,

$$\lim_{n \to \infty} z_i^{(n)} = z_i \quad \text{for all } i,$$

for all initial vectors $z^{(0)}$. 
Theorem 2.1

An iterative method which can be expressed in the form of equation (2.9) converges if and only if $\rho(M) < 1$.

Proof

If we define the error vector after $n$ iterations to be

$$ e^{(n)} = z^{(n)} - z $$  \hspace{1cm} (2.10)

and assume the method consistent, i.e.

$$ z = Mz + d $$  \hspace{1cm} (2.11)

we have from (2.9) and (2.10),

$$ e^{(n+1)} = Me^{(n)} $$  \hspace{1cm} (2.12)

and hence

$$ e^{(n)} = M^n e^{(0)} $$  \hspace{1cm} (2.13)

We require the conditions under which $e^{(n)} \to 0$ as $n \to \infty$. From equation (2.13), this can happen if and only if $M^n \to 0$ (the null matrix) as $n \to \infty$.

By Theorem 1.7, this will be true if and only if $\rho(M) < 1$. This completes the proof.

By assuming the conditions (2.4) on the matrix $A$, it can be shown (e.g. see Young and Frank (1963), p.15) that the Jacobi method (2.8) converges.

Practical determination of convergence

Since in practical computation, numbers can only be stored to a limited accuracy, some rounding error occurs. It is very unlikely, therefore, that the condition $z^{(n+1)} = z^{(n)}$ can be fulfilled in practice. Therefore we must seek some other method of determining when an iterative process has converged. The usual approach is to reduce the norm of the error vector $e^{(n)}$ to less than some predetermined factor $\varepsilon$ of the norm of the initial error vector $e^{(0)}$. 
From (2.13) we have

\[ ||\mathbf{e}^{(n)}|| = ||M^n\mathbf{e}^{(0)}|| \leq ||M^n|| \cdot ||\mathbf{e}^{(0)}||. \]  

(2.14)

Then, provided \( \mathbf{e}^{(0)} \) is not the null vector,

\[ ||\mathbf{e}^{(n)}|| / ||\mathbf{e}^{(0)}|| \leq ||M^n|| \]  \hspace{1cm} (2.14')

and we require

\[ ||\mathbf{e}^{(n)}|| \leq \varepsilon ||\mathbf{e}^{(0)}|| \]  \hspace{1cm} (2.15)

where \( ||.|| \) means \( ||.||_2 \) as defined in Chapter 1. By Theorem 1.7 we know that \( ||M^n|| \to 0 \) as \( n \to \infty \) if and only if \( \rho(M) < 1 \). Hence we can satisfy (2.15) by choosing \( n \) sufficiently large that

\[ ||M^n|| \leq \varepsilon. \]  \hspace{1cm} (2.16)

If \( n \) is large enough so that \( ||M^n|| < 1 \), this is equivalent to

\[ n \geq -\log \varepsilon / (-\frac{1}{n} \log ||M^n||). \]  \hspace{1cm} (2.16')

**Definition 2.3**

We now define

\[ R_n(M) = \frac{1}{n} \log ||M^n|| \]  \hspace{1cm} (2.17)

as the average rate of convergence (for \( n \) iterations) of the iterative method associated by (2.9) with the matrix \( M \).

**Definition 2.4**

We also define the asymptotic average rate of convergence, or asymptotic rate of convergence by

\[ R(M) = \lim_{n \to \infty} R_n(M) = -\log \rho(M). \]  \hspace{1cm} (2.18)

The latter equality holds, since

\[ \rho(M) = \lim_{n \to \infty} (||M^n||)^{1/n}, \]

a result proved by Young (1971), p.87. We shall refer to \( R(M) \) as the rate of convergence.
Young (ibid), p. 88, further proves that $||M^n||$ behaves as

$$\binom{n}{p-1} [\rho(M)]^{n-p+1},$$

(2.19)

where $\binom{n}{p-1}$ is the binomial coefficient and $p$ is the order of the largest diagonal block of the Jordan canonical form of $M$. In particular, if all the eigenvalues are distinct then $p=1$ and $||M^n||$ behaves like $[\rho(M)]^n$; if there are two eigenvalues of modulus $\rho(M)$, $p=2$ and $||M^n||$ behaves as $n[\rho(M)]^{n-1}$. Combining this with equation (2.16), gives, for $p=1$, the approximate relationship

$$[\rho(M)]^n \leq \varepsilon.$$

(2.20)

This can be rewritten as

$$n \geq \frac{\log \varepsilon}{\log [\rho(M)]}$$

(2.20')

$$n \geq \frac{-\log \varepsilon}{R(M)},$$

so that $n$ is an estimate of the number of iterations required for convergence. For the case where $p=2$, the smallest value of $n$ such that

$$n[\rho(M)]^{n-1} \leq \varepsilon$$

(2.21)

estimates the number of iterations required.

The process of testing whether the error norm has been reduced by a given factor cannot be applied directly, since of course the error vector is unknown. However, we can develop a close approximation based on the difference between successive iterates. We have

$$||z^{(n+1)} - z^{(n)}|| = ||(e^{(n+1)} - z) - (e^{(n)} - z)||$$

$$= ||e^{(n+1)} - e^{(n)}|| = ||M^n(M-I)e^{(0)}||$$

(2.22)

from (2.13). Hence

$$||e^{(n+1)} - e^{(n)}||/||e^{(0)}|| \leq ||M^n||.||M-I||$$

(2.23)

and so the reduction factor at the $(n+1)^{th}$ iteration is a (constant) multiple of $||M^n||$. 
The numerical experiments in this thesis do not use $\|z^{(n+1)} - z^{(n)}\|_2$, since the evaluation of this quantity is non-trivial. Instead, they use $\|z^{(n+1)} - z^{(n)}\|_\infty$, the maximum difference between successive iterates. The test
\[ \|z^{(n+1)} - z^{(n)}\|_\infty < \epsilon \]
is slightly less stringent than
\[ \|z^{(n+1)} - z^{(n)}\|_2 < \epsilon, \]
but this makes no significant difference in practice.

2.2.2 The Point Gauss-Seidel and SOR Methods

The iterative processes described in this sub-section are point successive methods. Using the splitting of the coefficient matrix $A$ defined by equation (2.7), and letting
\[ \hat{L} = D^{-1}L, \quad \hat{U} = D^{-1}U, \quad \xi = D^{-1}b \]
the Gauss-Seidel method is written
\[ z^{(n+1)} = \hat{L}z^{(n+1)} + \hat{U}z^{(n)} + \xi \]
or alternatively, in the form of (2.9),
\[ z^{(n+1)} = (I - \hat{L})^{-1} \hat{U}z^{(n)} + (I - \hat{L})^{-1} \xi. \]
Like the Jacobi method, this process can be shown to converge if the conditions (2.4) apply to the matrix $A$. (See e.g. Young and Frank (1963), pp.15-16).

The successive over-relaxation method (SOR method for short), proposed by Young (1954), pp.92-111, is an extrapolation of equation (2.25). It is written as
\[ z^{(n+1)} = (1 - \omega)z^{(n)} + \omega(\hat{L}z^{(n+1)} + \hat{U}z^{(n)} + \xi), \]
or
\[ z^{(n+1)} = M_\omega z^{(n)} + (I - \hat{L})^{-1} \xi, \]
where
\[ M_\omega = (I - \omega \hat{L})^{-1}(I - \omega \hat{L}I + \omega \hat{U}). \]
Here $\omega$ is a parameter known as a relaxation factor, the choice of which determines the rate of convergence of the method. When $\omega=1$, the SOR method reduces to the Gauss-Seidel method. Varga (1962), p. 80 proves that if $A$ (of order $N$) is symmetric and $a_{i,i} > 0$, $1 \leq i \leq N$, then $\rho(M_\omega) < 1$ (i.e. the SOR method converges) if and only if $A$ is positive definite and $0 < \omega < 2$.

**Comparison of rates of convergence**

Direct comparisons of the Jacobi, Gauss-Seidel and SOR methods can be made when the matrix $A$ has property (A) and is consistently ordered, as described in Chapter 1. We will assume that these two properties hold in any future discussion on the SOR method, unless otherwise stated.

We summarise here the SOR theory, which is given in detail in Young (1971), pp. 142-144 and 188-189.

Let $\omega$ be real and positive, then $\lambda \neq 0$ is an eigenvalue of the SOR matrix $M_\omega$ if and only if there exists an eigenvalue $\mu$ of the Jacobi matrix $B$ (see (2.8)) such that

$$\frac{\lambda + \omega - 1}{\omega^\delta} = \mu \quad (2.27)$$

If we denote $\rho(B)$ by $\bar{\mu}$, then it can be shown that

$$\omega_b = \frac{2}{1 + \frac{1}{1 - \bar{\mu}}^2} \quad (2.28)$$

is the value of $\omega$ which minimises $\rho(M_\omega)$. Using this optimum value of $\omega$, it can also be shown that

$$\rho(M_{\omega_b}) = \frac{1 + \sqrt{1 - \bar{\mu}^2}}{1 + \sqrt{1 - \bar{\mu}^2}} = \omega_b - 1 \quad (2.29)$$

For the Gauss-Seidel method ($\omega=1$), equation (2.27) gives

$$\rho(M_1) = [\rho(B)]^2 \quad (2.30)$$

and so from (2.18),

$$R(M_1) = 2R(B) \quad (2.31)$$

Thus the rate of convergence of the Gauss-Seidel method is twice that of the Jacobi method. By the use of (2.29), it can be verified that asymptotically as $\rho(B) \to 1$.
To compare the efficiency of iterative methods it is necessary to consider the computational effort for each iteration as well as the number of iterations required for convergence. One way of estimating this effort is to count the number of arithmetic operations performed per point. This gives a rather crude estimate, since different operations take different times to perform on a computer. For example, multiplications take longer than additions. Nevertheless, the estimate is sufficiently accurate to indicate whether or not there is a significant difference between the efficiency of two methods. In counting operations we include subtractions with additions, since they take about the same time to execute, but list multiplications and divisions separately.

For a matrix derived from a k-point finite-difference approximation, both the Jacobi and Gauss-Seidel methods need \((k-1)\) additions and \((k-1)\) multiplications per point, a total of \((2k-2)\) operations. Since the Gauss-Seidel method converges (asymptotically) twice as fast as the Jacobi, it is the more efficient method by a factor of two. The point SOR method needs \((k+1)\) additions and \(k\) multiplications, a total of \((2k+1)\) operations. From equations (2.32) and (2.20'), we see that the Gauss-Seidel method takes (asymptotically) \(\frac{2}{\sqrt{R(M_1)}}\) times as many iterations to converge as point SOR. Hence the latter method is superior provided it takes less than \(\frac{\sqrt{R(M_1)}}{2}\) times as much work per iteration as the Gauss-Seidel method, i.e. provided

\[
\frac{\sqrt{R(M_1)}}{2} (2k+2) < (2k-1).
\]

In practice this inequality almost always holds. We conclude, therefore, that point SOR is a more efficient iterative method than either the Gauss-Seidel or Jacobi methods.

For our model problem in the unit square \(m^2\) internal points), Varga (1962), pp.203-204 gives:

\[
R(M_0) = 2 \sqrt{R(M)}.
\]
Also the Jacobi and Gauss-Seidel methods take 4 additions and 4 multiplications per point per iteration and the SOR method takes 6 additions and 5 multiplications.

2.2.3 The Block Gauss-Seidel and SOR Methods

In point iterative methods, each component of the iterate \( z^{(n)} \) is determined *explicitly*, i.e. it can be calculated by itself using already computed approximate values of the other components. With block iterative methods, one improves the values of the approximate solution simultaneously on some block (group) of points in \( \mathbb{R} \). Such methods are called *implicit* methods since we solve a linear system for a whole group of components at once. The advantage of using implicit rather than explicit methods is that they may have appreciably larger convergence rates, at the cost of some extra computation in each iteration.

The twin objectives in choosing the form of block are to minimise the spectral radius of the iteration matrix and to minimise the computational effort of performing each iteration.

We illustrate three types of block on our model problem. The first is the column or line block, for which the points are ordered as in Figure 2.2. The second is the two-line block, for which we number the points as shown in Figure 2.3. The third is the peripheral block, for which we order the points spiral-wise as in Figure 2.4. In each case, the resulting coefficient matrix is block tridiagonal, and therefore has *block property* (A)
and block consistent ordering. These properties are the block analogues of (point) property (A) and consistent ordering as defined in Chapter 1; full definitions of them are given by Arms, Gates and Zondek (1956), pp.220-229. Hence we can derive a block form of SOR.

Let us assume for convenience that our model problem has an even number of columns, so that there are an exact number of two-line blocks. We have $m$ line blocks, $\frac{m}{2}$ two-line blocks and $\frac{m}{2}$ peripheral blocks.

In block iterative methods, it is necessary to solve sub-systems of equations of the form

$$A \frac{z}{r \rightarrow r-1} + B \frac{z}{r \rightarrow} + C \frac{z}{r \rightarrow r+1} = \frac{b}{r \rightarrow}$$

(2.34a)

for $B \frac{z}{r \rightarrow}$, then solve

$$B \frac{z}{r \rightarrow} = k \frac{z}{r \rightarrow}$$

(2.34b)

for $\frac{z}{r \rightarrow}$. The terms are defined as follows.

Line block: $1 \leq r \leq m$;

$$A \frac{r}{r \rightarrow}, B \frac{r}{r \rightarrow}, C \frac{r}{r \rightarrow}, 1 \leq r \leq m,$$ as in (2.5b) with $A_1 \frac{r}{r \rightarrow}, C_m \frac{r}{r \rightarrow} = 0$;

$\frac{z}{r \rightarrow}$ is the vector of values on the $r^{th}$ line.

(2.34c)

Two-line block: $1 \leq r \leq \frac{m}{2}$;

$$A \frac{r}{r \rightarrow}, B \frac{r}{r \rightarrow}, C \frac{r}{r \rightarrow}, 1 \leq r \leq \frac{m}{2},$$ as in (2.5c) with $A_1 \frac{r}{r \rightarrow}, C_m \frac{r}{r \rightarrow} = 0$;

$\frac{z}{r \rightarrow}$ is the vector of values on the $(2r-1)^{th}$ and $(2r)^{th}$ lines.

(2.34d)

Peripheral block: $1 \leq r \leq \frac{m}{2}$;

$$A \frac{r}{r \rightarrow}, B \frac{r}{r \rightarrow}, C \frac{r}{r \rightarrow}, 1 \leq r \leq \frac{m}{2},$$ as in (2.5d) with $A_1 \frac{r}{r \rightarrow}, C_m \frac{r}{r \rightarrow} = 0$;

$\frac{z}{r \rightarrow}$ is the vector of values on the $r^{th}$ peripheral.

(2.34e)

The block SOR method for solving equation (2.34) is obtained by solving

$$A \frac{z}{r \rightarrow r-1}^{(n+1)} + B \frac{z^*}{r \rightarrow}^{(n+1)} + C \frac{z}{r \rightarrow r+1}^{(n)} = \frac{b}{r \rightarrow}$$

(2.35a)
for $B \frac{z^*(n+1)}{r_x}$, then solving

$$B \frac{z^*(n+1)}{r_x} = k$$  \hspace{1cm} (2.35b)

for $\frac{z^*(n+1)}{r_x}$, and extrapolating

$$\frac{z(n+1)}{r_x} = \omega(\frac{z^*(n+1)}{r_x} - \frac{z(n)}{r_x}) + \frac{z(n)}{r_x}$$  \hspace{1cm} (2.35c)

to obtain the final solution.

The theory of block SOR, presented, for example, by Young (1971) pp.445-452, is similar to that of point SOR. Briefly, if we let $\mu(B)$ be the spectral radius of the block Jacobi matrix $B(B)$ then the optimum value of the SOR parameter is given by

$$\omega_b(B) = \frac{2}{1 + \sqrt{1 - (\mu(B))^2}}$$ \hspace{1cm} (2.36)

the spectral radius of the SOR matrix $M(B)$ by

$$\lambda(B) = \omega_b(B) - 1$$ \hspace{1cm} (2.37)

and the rate of convergence of SOR, asymptotically as $\mu(B)^{-1}$, by

$$R(M(B)) = 2 \sqrt{R(M_{\omega=1}(B))} \left( \frac{R(B)}{\omega_b(B)} \right)$$ \hspace{1cm} (2.38)

Varga (1962) pp.204-205, shows for the model problem, the rate of convergence of the line method (SLOR), is

$$R(M_{\omega_b}(B)) = \frac{2 \sqrt{2\pi}}{m+1}, \ m \rightarrow \infty$$ \hspace{1cm} (2.39)

and that of the two-line method (S2LOR) is

$$R(M_{\omega_b}(B)) = \frac{4\pi}{m+1}, \ m \rightarrow \infty$$ \hspace{1cm} (2.40)

Benson (1969) p.268 states that the rate of convergence (for the model problem) of the peripheral method (SPOR) is also

$$R(M_{\omega_b}(B)) = \frac{4\pi}{m+1}, \ m \rightarrow \infty$$ \hspace{1cm} (2.41)
Efficient algorithms exist for solving equation (2.35b). In the SLOR method the matrices $B_i$ have the form

\[
\begin{bmatrix}
  b_1 & -c_1 & 0 & \cdots & 0 \\
  -a_2 & b_2 & -c_2 & \cdots & 0 \\
  0 & \ddots & \ddots & \ddots & \vdots \\
  0 & \cdots & 0 & b_{N-1} & -c_{N-1} \\
  0 & \cdots & 0 & -a_N & b_N \\
\end{bmatrix}
\] (2.42)

A well-known algorithm for the problem (based on Gaussian elimination) is as follows:

We compute

\[
\begin{align*}
g_1 &= \frac{c_1}{b_1} \\
f_1 &= \frac{k_1}{b_1} \\
\end{align*}
\]

\[
\begin{align*}
g_i &= \frac{c_i}{b_i - a_i g_{i-1}} \\
f_i &= \frac{k_i + a_i f_{i-1}}{b_i - a_i g_{i-1}}, \quad i = 2, 3, \ldots, N. \\
\end{align*}
\] (2.43a)

which transforms the matrix equation to

\[
\begin{bmatrix}
  1 & -g_1 & 0 & \cdots & 0 \\
  1 & -g_2 & 0 & \cdots & 0 \\
  0 & \ddots & \ddots & \ddots & \vdots \\
  0 & \cdots & 0 & 1 & -g_{N-1} \\
  0 & \cdots & 0 & 1 & 1 \\
\end{bmatrix}
\begin{bmatrix}
  z_1 \\
  z_2 \\
  \vdots \\
  z_{N-1} \\
  z_N \\
\end{bmatrix}
= \begin{bmatrix}
  f_1 \\
  f_2 \\
  \vdots \\
  f_{N-1} \\
  f_N \\
\end{bmatrix}.
\] (2.43b)

The solution $z$ is then found by the back substitution

\[
\begin{align*}
z_N &= f_N; \\
z_i &= f_i + g_i z_{i+1}, \quad i = N-1, N-2, \ldots, 1.
\end{align*}
\] (2.43c)
Provided the matrix does not change, the g's need only be calculated once, and we can set

\[ d_1 = \frac{1}{b_1} \]

\[ d_i = \frac{1}{b_i - a_i g_{i-1}} , \text{ i}=2,3,\ldots,N. \]  \hspace{1cm} (2.43d)

Thus the solution of equation (2.35b) for SLOR is found by the forward substitution

\[ f_1 = k_1 d_1 \]

\[ f_i = (k_i + a_i f_{i-1}) d_i , \text{ i}=2,3,\ldots,N \]

followed by (2.43c). (We use the inverses of the \( b_i - a_i g_{i-1} \) terms because multiplication is a faster computer operation than division). Solving the block by this method requires 2 additions and 3 multiplications per point. If \( B \) is symmetric and positive definite, the normalised triangular decomposition algorithm of Cuthill and Varga (1959), pp.236-244 can be used. This algorithm needs only 2 additions and 2 multiplications per point. By considering equations (2.35a) and (2.35c) as well as (2.35b), it can be shown that SLOR applied to a matrix derived from a k-point finite-difference approximation requires \((k+1)\) additions and \(k\) multiplications per point per iteration, provided the block matrices are tridiagonal, symmetric and positive definite. This is exactly the same as for point SOR. Since for the model problem, the rate of convergence of SLOR is asymptotically \( \sqrt{2} \) greater, it is clearly the more efficient method of the two.

In S2LOR, the matrices \( B \) have the form
Equation (2.35b) can be transformed by a Gaussian elimination process to the form

\[
\begin{bmatrix}
1 & -\beta_1 & -\gamma_1 \\
1 & -\beta_2 & -\gamma_2 \\
1 & -\beta_3 & -\gamma_3 \\
0 & & \\
0 & & \\
\end{bmatrix}
\begin{bmatrix}
z_1 \\
z_2 \\
z_3 \\
z_{N-2} \\
z_{N-1} \\
z_N \\
\end{bmatrix}
= \begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
f_{N-2} \\
f_{N-1} \\
f_N \\
\end{bmatrix}
\]

(2.45a)

where the matrix entries are defined as follows using the auxiliary vectors \( \delta \) and \( \omega \). We have

\[
\omega_1 = \frac{1}{c_1}, \quad \beta_1 = d_1 \omega_1, \quad \gamma_1 = e_1 \omega_1;
\]

\[
\delta_2 = b_2, \quad \omega_2 = \frac{1}{c_2 - \delta_2 b_1}, \quad \beta_2 = (d_2 + \delta_2 \gamma_1) \omega_2, \quad \gamma_2 = e_2 \omega_2;
\]

then

\[
\delta_i = b_i + a_i \beta_{i-2},
\]

\[
\omega_i = \frac{1}{c_i - \delta_i \beta_{i-1} - a_i \gamma_{i-2}},
\]

\[
\beta_i = (d_i + \delta_i \gamma_{i-1}) \omega_i,
\]

\[
\gamma_i = e_i \omega_i, \quad i = 3, 4, \ldots, N
\]
except that

$$\beta_{N-1} \gamma_{N-1} \gamma_N = 0.$$  

(2.45b)

The right-hand side $f$ is computed by the forward substitution

$$f_1 = k_1 \omega_1 ;$$

$$f_2 = (k_2 + \delta_2 f_1) \omega_2 ;$$

then

$$f_i = (k_i + \delta_i f_{i-1}) \omega_i, \quad i=3,4,\ldots,N$$

and the solution $z$ found by the back substitution process

$$z_N = f_N ;$$

$$z_{N-1} = f_{N-1} + \beta_{N-1} z_N ;$$

$$z_i = f_i + \beta_i z_{i+1} + \gamma_i z_{i+2}, \quad i=N-2,N-1,\ldots,1.$$  

(2.45c)

The solution process for this block requires 4 additions and 5 multiplications per point. As for the tridiagonal block of SLOR, a more efficient method, normalised triangular decomposition, can be used if the matrix is symmetric and positive definite. This algorithm presented by Benson (1969), pp.91-95, requires 4 additions and 4 multiplications per point. Using this method, S2LOR applied to the model problem requires 7 additions and 6 multiplications per point per iteration, 2 more operations than SOR and SLOR. However, its rate of convergence is asymptotically $\sqrt{2}$ greater than that of SLOR, and hence is a more efficient method than either. (If there are an odd number of mesh lines parallel to both axes the S2LOR method fails to incorporate one line of points, which can be solved by SLOR. This does not significantly affect our conclusions about the S2LOR method stated above).

For SPOR, the matrices $B_r$ have the form
Benson (1969), pp. 96-103 and 108-111 presents three algorithms for solving this block: Gaussian elimination, LU decomposition and, for the case when \( B \) is symmetric and positive definite, a normalised triangular decomposition algorithm. He calls them 'periodic tridiagonal' algorithms because such systems of equations also occur for the line ordering of points when the boundary conditions are periodic.

The Gaussian elimination algorithm transforms the matrix equation (2.35b) to the form

\[
\begin{bmatrix}
1 & -g_1 & & & & \\
& 1 & -g_2 & & & \\
& & 0 & & & \\
& & & 1 & -g_{N-2} & \\
& & & & 0 & 1 \\
& & & & & 1 \\
\end{bmatrix}
\begin{bmatrix}
z_1 \\
z_2 \\
z_3 \\
z_4 \\
z_5 \\
z_N \\
\end{bmatrix}
= 
\begin{bmatrix}
f_1 \\
f_2 \\
f_3 \\
f_4 \\
f_5 \\
f_N \\
\end{bmatrix}
\]

(2.47a)

For convenience in computing the \( g \)'s, \( h \)'s and \( f \)'s, we use the extra vectors \( d, F, G \) and \( H \). To determine the entries in the above matrix we first evaluate

\[
d_1 = \frac{1}{b_1}, \quad g_1 = c_1 d_1, \quad h_1 = a_1 d_1,
\]

\[
G_1 = c_N, \quad H_1 = b_N.
\]
We then compute recursively

\[ d_i = \frac{1}{b_i - a_i g_{i-1}}, \]

\[ g_i = c_i d_i, \]

\[ h_i = a_i h_{i-1} d_i, \]

\[ G_i = g_{i-1} G_{i-1}, \]

\[ H_i = H_{i-1} - G_{i-1} h_{i-1}, \quad i = 2, 3, \ldots, N-1, \]

and finally we evaluate

\[ d_N = \frac{1}{(H_{N-1} - (G_{N-1} + a_N)(g_{N-1} + h_{N-1}))}. \quad (2.47b) \]

Provided the matrix does not change, this need only be calculated once.

The \( f \) and \( F \) terms are found by the forward substitutions

\[ f_1 = k_1 d_1, \]

\[ F_1 = k_N, \]

\[ f_i = (h_i + a_i f_{i-1}) d_i, \]

\[ F_i = F_{i-1} + G_{i-1} f_{i-1}, \quad i = 2, 3, \ldots, N-1; \]

\[ f_N = (F_{N-1} + (G_{N-1} + a_N) f_{N-1}) d_N. \]

The solution \( z \) is then computed by back substitution as follows:

\[ z_N = f_N, \]

\[ z_i = f_i + g_i z_{i+1} + h_i z_N, \quad i = N-1, N-2, \ldots, 1 \quad (2.47c) \]

The solution of the block by this method requires 4 additions and 5 multiplications per point, and the LU algorithm requires the same. The normalised algorithm needs only 4 additions and 4 multiplications per point. Taking into consideration also equations (2.35a) and (2.35c) it can be shown that SPOR applied to a matrix derived from a \( k \)-point finite-difference approximation requires \((k+3)\) additions and \((k+2)\) multiplications.
per point per iteration, provided the block matrices are periodic tridiagonal, symmetric and positive definite. Since the rate of convergence of SPOR is asymptotically $\sqrt{2}$ greater than that of SLOR, we see that the two methods are of roughly comparable efficiency for the model problem (k=5). It is not possible to make any stronger statement about their relative merits without making a more detailed appraisal of the work per iteration than a simple count of arithmetic operations.

2.2.4 Experimental Results for SOR on the Model Problem

The experimental results presented here are for our model problem, Laplace's equation in the unit square, with the boundary conditions $t(x,y)=0$. The initial estimate $z^{(0)}$ of the solution has all elements unity in every case.

The spectral radii $\rho(B)$ of the point, line, two-line and peripheral Jacobi iteration matrices are obtained experimentally by means of the power method. The results obtained for the point and line matrices agree closely with the theoretical values

$$\rho(B) = \cos \pi h$$

(2.48)

for the point case and

$$\rho(B) = \frac{\cos \pi h}{2-\cos \pi h}$$

(2.49)

for the line case, where $h$ is the mesh size. No theoretical values are known for the two-line and peripheral matrices. The theoretical optimum values of $\omega$ are obtained from equation (2.28) using the values of $\rho(B)$ as found by experiment, and the spectral radii of the SOR iteration matrices from equation (2.29). Using this information, the estimated number of iterations required for convergence is found from (2.20') using $\epsilon=5 \times 10^{-6}$. The experimental optimum values of $\omega$ are determined to within 0.01 by solving the problems for a range of values of $\omega$ and choosing those which give the minimum number of iterations.
### TABLE 2.1

Laplace's equation in the unit square, Point SOR

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td>5</td>
<td>0.809017</td>
<td>$\frac{5}{9}$</td>
<td>12.3</td>
</tr>
<tr>
<td>10</td>
<td>0.951056</td>
<td>19</td>
<td>25.1</td>
</tr>
<tr>
<td>15</td>
<td>0.978147</td>
<td>29</td>
<td>37.2</td>
</tr>
<tr>
<td>20</td>
<td>0.987688</td>
<td>39.4</td>
<td>50.5</td>
</tr>
<tr>
<td>25</td>
<td>0.992114</td>
<td>48</td>
<td>60.5</td>
</tr>
<tr>
<td>30</td>
<td>0.994522</td>
<td>58</td>
<td>74.5</td>
</tr>
<tr>
<td>35</td>
<td>0.995974</td>
<td>68</td>
<td>80.5</td>
</tr>
<tr>
<td>40</td>
<td>0.996917</td>
<td>78</td>
<td>90.5</td>
</tr>
</tbody>
</table>
TABLE 2.2
Laplace's equation in the unit square. Line SOR

<table>
<thead>
<tr>
<th>h^{-1}</th>
<th>\rho(B)</th>
<th>Iterations</th>
<th>Optimum \omega</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td>5</td>
<td>0.679285</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>0.906680</td>
<td>14</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>0.957230</td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td>20</td>
<td>0.975676</td>
<td>27</td>
<td>38</td>
</tr>
<tr>
<td>25</td>
<td>0.984353</td>
<td>34</td>
<td>47</td>
</tr>
<tr>
<td>30</td>
<td>0.989103</td>
<td>41</td>
<td>55</td>
</tr>
<tr>
<td>35</td>
<td>0.991981</td>
<td>48</td>
<td>65</td>
</tr>
<tr>
<td>40</td>
<td>0.993854</td>
<td>55</td>
<td>75</td>
</tr>
</tbody>
</table>
TABLE 2.3
Laplace's equation in the unit square. Two-line SOR

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td>5</td>
<td>0.509646</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>0.829982</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>0.917813</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>0.952521</td>
<td>19</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>0.969172</td>
<td>24</td>
<td>30</td>
</tr>
<tr>
<td>30</td>
<td>0.978443</td>
<td>29</td>
<td>38</td>
</tr>
<tr>
<td>35</td>
<td>0.984086</td>
<td>34</td>
<td>43</td>
</tr>
<tr>
<td>40</td>
<td>0.987783</td>
<td>39</td>
<td>49</td>
</tr>
</tbody>
</table>
TABLE 2.4

Laplace's equation in the unit square. Peripheral SOR

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td>5</td>
<td>0.632456</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>0.897090</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>0.956477</td>
<td>20</td>
<td>22</td>
</tr>
<tr>
<td>20</td>
<td>0.975054</td>
<td>27</td>
<td>29</td>
</tr>
<tr>
<td>25</td>
<td>0.984253</td>
<td>34</td>
<td>37</td>
</tr>
<tr>
<td>30</td>
<td>0.988979</td>
<td>41</td>
<td>44</td>
</tr>
<tr>
<td>35</td>
<td>0.991955</td>
<td>48</td>
<td>51</td>
</tr>
<tr>
<td>40</td>
<td>0.993814</td>
<td>55</td>
<td>59</td>
</tr>
</tbody>
</table>

|          |             | Theory     | Experiment       |
| 1.13     | 1.13-1.17   |
| 1.39     | 1.42        |
| 1.55     | 1.56        |
| 1.64     | 1.65        |
| 1.70     | 1.71        |
| 1.74     | 1.75        |
| 1.78     | 1.78        |
| 1.80     | 1.80-1.81   |
In all four tables, the theoretical optimum values of $\omega$ agree very closely with the experimental ones. However, the agreement between theory and experiment in number of iterations is more variable; it is excellent for SPOR, moderate for point SOR, slightly worse for S2LOR and poor for SLOR.

In each of these methods the iteration matrix $M_\omega$ has two eigenvalues whose magnitude is the spectral radius and hence $||M_\omega||$ behaves not as $\rho(M_\omega)^n$ but as $n[\rho(M_\omega)]^{n-1}$, as described earlier. Hence, strictly-speaking, equation (2.21) and not equation (2.20') should be used to estimate the number of iterations required. Table 2.5 shows the results of applying the modified theory.

The modified theory gives excellent results for SLOR, good ones for S2LOR, moderate ones for point SOR and very poor ones for SPOR. It appears that the extent to which the twin eigenvalues of largest modulus affect the convergence depends very much on the method. Intuitively this is feasible, although it is at variance with well-established theory, and no satisfactory explanation has been found.

A comparison of the actual number of iterations needed by the four methods shows that point SOR takes about 1.3 times as many iterations as SLOR, 1.6 times as many as SPOR and 1.9 times as many as S2LOR. These ratios are well below their theoretical asymptotic values of $\sqrt{2}$, 2 and 2 respectively, which is partly due to the varying effects of the twin eigenvalues of largest modulus. The corresponding ratios for the estimated number of iterations (using either theory) are 1.4 for SLOR, 1.4 for SPOR, and 2.0 for S2LOR. The figures for the line and two-line methods accord excellently with the theory, but indicate that the theoretical rate of convergence of the peripheral method is only a factor of $\sqrt{2}$ faster than the point method, not a factor of 2. (Indeed, the factor of 2 difference has not been rigorously proved).
| $h^{-1}$ | **Point SOR** | | **SLOR** | | **S2LOR** | | **SPOR** |
|---|---|---|---|---|---|---|
| 5 | 12 | 12 | 9 | 9 | 7 | 7 | 8 | 8 |
| 10 | 26 | 25 | 18 | 20 | 13 | 13 | 17 | 15 |
| 15 | 39 | 37 | 28 | 28 | 20 | 18 | 27 | 22 |
| 20 | 53 | 50 | 37 | 38 | 26 | 25 | 36 | 29 |
| 25 | 67 | 60 | 47 | 47 | 33 | 30 | 46 | 37 |
| 30 | 81 | 74 | 56 | 55 | 39 | 38 | 56 | 44 |
| 35 | 95 | 80 | 66 | 65 | 46 | 43 | 66 | 51 |
| 40 | 109 | 90 | 76 | 75 | 53 | 49 | 76 | 59 |
2.2.5 Methods of Simultaneous Iteration

The Jacobi method described in sub-section 2.2.1 can be extrapolated to give a faster rate of convergence when the coefficient matrix \( A \) (of order \( N \)) is real and positive definite. We assume without loss of generality that all the diagonal entries \( a_{ii} \) of \( A \) are unity, so that

\[
A = I - L - U,
\]

(2.50)

where \( -L \) and \( -U \) are the strictly lower triangular and strictly upper triangular components of \( A \) respectively. The Jacobi method for the solution of equation (2.3) is then written as

\[
z^{(n+1)} = (L + U)z^{(n)} + b, \quad n \geq 0
\]

(2.51)

or alternatively

\[
z^{(n+1)} = (I - A)z^{(n)} + b = z^{(n)} + \Delta z^{(n)} ,
\]

(2.51')

which shows that the change in the solution vector is equal to the residual vector \( \Delta z^{(n)} = b - Az^{(n)} \). Equation (2.51') can be extrapolated by multiplying \( \Delta z^{(n)} \) by a constant \( a \) or a variable \( a_n \), giving rise to the iterative methods

\[
z^{(n+1)} = (I - aA)z^{(n)} + ab,
\]

(2.52)

called the simultaneous displacement method and

\[
z^{(n+1)} = (I - \alpha_n A)z^{(n)} + \alpha_n b
\]

(2.53)

called Richardson's method.

For the simultaneous displacement method, we require

\[
\rho(I - \alpha A) < 1
\]

(2.54)

for convergence. If we denote the eigenvalues of \( A \) by \( \lambda_i \), \( 1 \leq i \leq N \), where

\[
\lambda_1 = \max_i \lambda_i
\]

(2.55a)

and

\[
\lambda_N = \min_i \lambda_i
\]

(2.55b)

then the condition (2.54) can be written as

\[
|1 - a\lambda_i| < 1, \quad 1 \leq i \leq N.
\]

(2.56)
which gives the permissible range of values of $\alpha$ to be

$$0 < \alpha < \frac{2}{\lambda_1} \quad \text{(2.57)}$$

The fastest rate of convergence is obtained by choosing $\alpha$ such that $\rho(I-\alpha A)$ is minimised. Clearly the best choice of $\alpha$ is the one for which

$$1-\alpha \lambda_N = -(1-\alpha \lambda_1),$$

i.e.

$$\alpha = \frac{2}{\lambda_1 + \lambda_N} \quad \text{(2.58)}$$

With this choice of $\alpha$, $\rho(I-\alpha A)$ is given by

$$|1-\alpha \lambda_1| = |1-\alpha \lambda_N| = \frac{\lambda_1 - \lambda_N}{\lambda_1 + \lambda_N} = \frac{\rho - 1}{\rho + 1}, \quad \text{(2.59)}$$

where $\rho = \frac{\lambda_1}{\lambda_N}$ is the $P$-condition number of $A$, defined as the ratio of the maximum eigenvalue to the minimum eigenvalue of a positive definite matrix.

From equation (2.59) we obtain the rate of convergence

$$R(I-\alpha A) = \frac{2}{\rho}, \quad \rho \to \infty \quad \text{(2.60)}$$

In Richardson's method, the iteration matrix changes at each iteration and so the asymptotic rate of convergence of the method cannot be found directly. From equations (2.12) and (2.53) we have

$$e^{(n+1)} = \frac{1}{\|I-\alpha A\|} \sum_{i=0}^{n} (I-\alpha A)^i e^{(0)}$$

$$= Q^{n+1}(A)e^{(0)}, \quad \text{say.} \quad \text{(2.61)}$$

If the eigenvalues of $A$ are $\lambda_i$, $1 \leq i \leq N$ as before, the corresponding eigenvectors $v_i$ form a basis for the space, (2.61) can be written as

$$e^{(n+1)} = \sum_{i=1}^{N} a_i Q^{n+1}(A)v_i = \sum_{i=1}^{N} a_i Q^{n+1}(\lambda_i)v_i \quad \text{(2.61\text{'}},$$

where $e^{(0)} = \sum_{i=1}^{N} a_i v_i$, the $a_i$ being arbitrary.
For the error vector $e^{(n+1)}$ to be small, $Q_{n+1}(x)$ must be small for $x \in [\lambda_N, \lambda_1]$ subject to the constraint $Q_{n+1}(0)=1$. Such a polynomial has been given by Markoff (1916), pp. 213-258, and is

$$Q_{n+1}(x) = \frac{T_{n+1} \left[ \frac{\lambda_1 + \lambda_N - 2x}{\lambda_1 - \lambda_N} \right]}{T_{n+1} \left[ \frac{\lambda_1 + \lambda_N}{\lambda_1 - \lambda_N} \right]}$$  \hspace{1cm} (2.62a)

where

$$T_{n+1}(y) = \cos[(n+1)\cos^{-1}y] = \cosh[(n+1)\cosh^{-1}y]$$  \hspace{1cm} (2.62b)

is the Chebyshev polynomial of degree $(n+1)$. The $\alpha_i$ are chosen so that the roots of $\prod_{i=0}^{n} (1-\alpha_i x)$ are coincident with the roots of $T_{n+1}[(\lambda_i + \lambda_N - 2x)/(\lambda_i - \lambda_N)]$. It is necessary to choose the number $(n+1)$ of $\alpha_i$ in advance, and they may be quite large, causing unacceptable rounding errors.

To determine the average rate of convergence, we seek the maximum value of $Q_{n+1}(x)$,

$$\max_{a \leq x \leq b} |Q_{n+1}(x)| = \left[ T_{n+1}(w) \right]^{-1}$$  \hspace{1cm} (2.63)

where

$$w = \frac{\lambda_1 + \lambda_N}{\lambda_1 - \lambda_N}.$$

Using the relationship

$$2T_{n+1}(y) = [y+\sqrt{y^2-1}]^{n+1} + [y-\sqrt{y^2-1}]^{n+1}, \quad y>1,$$

(2.63) simplifies to

$$\max_{\lambda_N \leq x \leq \lambda_1} |Q_{n+1}(x)| \leq \frac{2}{[w+\sqrt{(w^2-1)}]^{n+1}}$$  \hspace{1cm} (2.64)

and it follows that the eigenvalues $Q_{n+1}(\lambda_i)$ of $Q_{n+1}(A)$ are uniformly bounded as $n \to \infty$. The asymptotic bound to the average rate of convergence is given by

$$\lim_{n \to \infty} \ln \left\{ \frac{2}{[w+\sqrt{(w^2-1)}]^{n+1}} \right\}$$
\[
\lim_{n \to \infty} \left\{ -\frac{2}{n+1} + \ln[w^2/(w^2-1)] \right\} = \ln[w^2/(w^2-1)]. \tag{2.65}
\]

For substantially large problems,\\n\[\omega = 1 + \frac{2}{p}\]
which gives the rate of convergence\\n\[R(\text{Richardson's method}) = \frac{2}{\sqrt{p}}. \tag{2.66}\]

Well-known methods exist which use the two previous iterates at each stage. The second-order Richardson's method, for example, is defined by\\n\[Z^{(n+1)} = Z^{(n)} + z(Z^{(n)} - Z^{(n-1)}) + a(Z^{(n)} - Z^{(n-1)}). \tag{2.67}\]

The values of \(a\) and \(b\) which maximise the rate of convergence of this method are proved by Frankel (1950), pp.65-75 to be\\n\[a = \left(\frac{2}{\sqrt{\lambda_1 + \sqrt{\lambda_N}}} \right)^2 \quad \text{and} \quad b = \left(\frac{1}{\sqrt{\lambda_1 + \sqrt{\lambda_N}}} \right)^2. \tag{2.68}\]

The spectral radius of the method is \(\sqrt{b}\) and the rate of convergence is again\\n\[R(\text{second-order Richardson}) \approx \frac{2}{\sqrt{p}}. \tag{2.69}\]

Note that this improvement in the rate of convergence over the simultaneous displacement method has been achieved at the expense of a 50% increase in the computer storage required.

The Chebyshev acceleration of (2.67) has the form\\n\[Z^{(n+1)} = Z^{(n)} + \alpha_n (Z^{(n)} - Z^{(n-1)}) + \beta_n (Z^{(n)} - Z^{(n-1)}), \tag{2.70}\]
where the parameters \(\alpha_n\) and \(\beta_n\) are defined by Stiefel (1958) pp.1-22 to be of the form\\n\[\alpha_n = \frac{4T_n(w)}{(\lambda_1 - \lambda_N)T_{n+1}(w)} \quad \text{and} \quad \beta_n = \frac{T_{n-1}(w)}{T_{n+1}(w)}. \tag{2.71}\]
Although this method requires more computer storage than Richardson's method, there is no need to pre-select the number of iterations; also the $\alpha_i$ and $\beta_i$ are small so that rounding errors are acceptably small. Since the minimising polynomial for this iteration is the same as for Richardson's method, the rate of convergence is also the same, i.e.

$$R(\text{Chebyshev acceleration}) \approx \frac{2}{\sqrt{P}}.$$

(2.72)

The Chebyshev semi-iterative method, presented by Golub and Varga (1961), pp.147-156, can be derived from (2.70) if we assume that $\lambda_1 = 1+\mu$, $\lambda_N = 1-\mu$ where $\mu$ is the spectral radius of the Jacobi method. The recurrence relation for the Chebyshev polynomials becomes

$$\alpha_n = 1+\beta_n \quad (2.73)$$

and equation (2.70) becomes

$$z^{(n+1)} = z^{(n)} + \alpha_n (b-Az^{(n)}) + (\alpha_n - 1)(z^{(n)} - z^{(n-1)}),$$

(2.74)

which simplifies to

$$z^{(n+1)} = \alpha_n [b+(L+U)z^{(n)} - z^{(n-1)}] + z^{(n-1)}.$$  

(2.74')

The parameters $\alpha_i$ are defined by

$$\alpha_1 = 1; \quad \alpha_2 = \frac{2}{2-\mu}; \quad \alpha_n+1 = \frac{1}{(1-\mu)\alpha_n/4}, \quad n>1.$$  

(2.75)

The rate of convergence is the same as for Chebyshev acceleration, i.e.

$$R(\text{Chebyshev semi-iterative method}) \approx \frac{2}{\sqrt{P}}.$$  

(2.76)

2.2.6 Preconditioning

For the methods described in the previous sub-section, it is clear that the optimal speed of convergence is limited by the value of $P$. It is possible to "precondition" the original finite-difference equations to reduce $P$ and so increase the rates of convergence of the methods.
Let \( \tilde{y} \) be an intermediate transformation vector given by
\[
\tilde{y} = (I-wU)z,
\]
where \( w \) is an acceleration parameter to be defined later. If we premultiply the original matrix equation
\[
Az = b \tag{2.78}
\]
by \((I-wL)^{-1}\), we obtain the equivalent expression,
\[
(I-wL)^{-1}A(I-wU)^{-1}[(I-wU)\tilde{z}] = (I-wL)^{-1}b \tag{2.79}
\]
which can be written as
\[
G^T A G \tilde{y} = d \tag{2.80}
\]
or
\[
B \tilde{y} = d \tag{2.80'}
\]
where \( G = (I-wU)^{-1} \) and \( d = (I-wL)^{-1}b \).

It is not necessary to calculate \( G \) explicitly in order to evaluate \( B \tilde{y} \), because an equation of the form
\[
\tilde{x} = G\tilde{y} \tag{2.81}
\]
where \( \tilde{x} \) is a known vector and \( G^{-1} \) a known upper triangular matrix, can be solved by back substitution. If \( G \) is replaced by \( G^T \), then since \((G^T)^{-1}\) is a known lower triangular matrix, the equation can be solved by forward substitution. Hence \( B \tilde{y} \) can be calculated efficiently.

Since \( A \) is a real positive definite and therefore symmetric matrix, it has a real positive definite square root \( A^\frac{1}{2} \) which is also symmetric (from Theorem 1.4). Since \( G \) and therefore \( A^\frac{1}{2}G \) is non-singular, it follows from Theorem 1.5 that
\[
(A^\frac{1}{2}G)^T(A^\frac{1}{2}G) = G^T A G \tag{2.82}
\]
is a real positive definite symmetric matrix.

The optimum acceleration parameter \( w_0 \) is defined as that value of \( w \) for which the ratio of largest to smallest eigenvalue \( \lambda_1(w)/\lambda_N(w) \) of \( G^T A G \) is minimised. If we let \( T = (I-wU)^{-1}(I-wL)^{-1}A \), then by the similarity transformation
\[(I-\omega U)T(I-\omega U)^{-1} = G^T AG, \quad (2.83)\]

\(T\) has the same eigenvalues, \(\lambda_i(\omega)\), \(1 \leq i \leq N\) as \(G^T AG\) but different eigenvectors. Then
\[(I-\omega U)^{-1}(I-\omega L)^{-1}A_y = \lambda_i(\omega)A_y, \quad 1 \leq i \leq N \quad (2.84)\]

where the \(A_y\), \(1 \leq i \leq N\) are the eigenvectors of \(T\). Premultiplying equation (2.84) by \(u_i^T(I-\omega L)(I-\omega U)\) gives
\[
\lambda_i(\omega) = \frac{\tau_i}{(1-\omega+\omega \tau_i + \omega^2 k_i)}, \quad (2.85)
\]

where \(\tau_i = u_i^T A u_i\) and \(k_i = u_i^T L u_i\). Hence the P-condition number of \(G^T AG\) is defined by
\[
P(\omega) = \frac{\tau_1(1-\omega+\omega \tau_N + \omega^2 k_N)}{\tau_N(1-\omega+\omega \tau_1 + \omega^2 k_1)}, \quad (2.86)
\]

For the function \(P(\omega)\) to possess a minimum, we must have
\[
\frac{d[P(\omega)]}{d\omega} = 0. \quad (2.87)
\]

If \(\tau_1, \tau_N, k_1\) and \(k_N\) are independent of \(\omega\), then equations (2.86) and (2.87) give
\[
c\omega^2 + d\omega + f = 0 \quad (2.88a)
\]

where
\[
c = k_N \tau_1 - k_1 \tau_N + k_N + k_1, \quad d = 2(k_N - k_1), \quad f = \tau_N - \tau_1. \quad (2.88b)
\]

Hence the optimum preconditioning parameter is given by
\[
\omega_b = \frac{(\tau_N - \tau_1)}{(k_1 - k_N)(\tau_N - \tau_1)^2 - (\tau_N - \tau_1)(k_N \tau_1 - k_1 \tau_N - k_N + k_1)} \quad (2.89)
\]

and the minimum P-condition number by equation (2.86) with \(\omega = \omega_b\). The eigenvalue spectrum of the matrix \(G^T AG\) is given by
\[
0 < \frac{\tau_N}{1-\omega_b + \omega_b \tau_N + \omega_b^2 k_N} \leq \lambda_i(\omega_b) \leq \frac{\tau_1}{1-\omega_b + \omega_b \tau_1 + \omega_b^2 k_1} < \infty, \quad 1 \leq i \leq N. \quad (2.90)
\]

If \(\tau_1, \tau_N, k_1\) and \(k_N\) are not independent of \(\omega\), expressions for \(\omega_b, P(\omega_b)\) and the eigenvalue spectrum can still be found. Details are given by Evans (1973), pp. 106-135.
From equation (2.83), it is clear that the preconditioning of equation (2.78) can be written as

\[ T \tilde{z} = (I - \omega U)^{-1} (I - \omega L)^{-1} A \tilde{z} = (I - \omega U)^{-1} (I - \omega L)^{-1} b \]  

(since any properties of the eigenvalues of \( G^T A G \) apply also to the matrix \( T \)) and this obviates the need to work with a transformed vector.

Evans (1968), pp. 295-314 considers in detail the application of preconditioning to our model problem and proves the following results.

**Theorem 2.2**

For all matrices \( A = I - L - U \) which have the form

\[ A = \begin{bmatrix} I & -U^* \\ -L^* & I_2 \end{bmatrix} \]

where \( L^* = U^T \) (this is called the \( \sigma_1 \) ordering) and possess property (A), the P-condition number of \( B_\omega = (I - \omega L)^{-1} A (I - \omega U)^{-1} \) is minimised when the parameter \( \omega \) is unity.

**Theorem 2.3**

For the model problem with points ordered column-wise (which gives a coefficient matrix with \( \sigma_2 \) ordering), \( \lambda_1(\omega) \) is minimised when \( \omega = 1 \). Further, \( \lambda_1(\omega) \) is monotonically decreasing for \( 0 < \omega < 1 \) and monotonically increasing for \( 1 < \omega < 2 \). \( \lambda_N(\omega) \) is maximised for some \( 1 < \omega < 2 \).

**Theorem 2.4**

For the model problem with points ordered column-wise, there is a value of \( \omega \) in the range \( 1 < \omega < 2 \) for which the P-condition number of \( B_\omega \) is minimised.

Evans (1973), pp. 106-135 gives experimental results for preconditioning a number of different problems. In all cases the minimum P-condition number \( P(\omega_b) \) of \( B_\omega \) is \( O(\sqrt{P(0)}) \), where \( P(0) \) is the P-condition number of \( A \). Actual values of \( P(\omega_b) \) range from about \( 0.5/\sqrt{P(0)} \) to about \( 2/\sqrt{P(0)} \) depending on the problem.
2.2.7 Preconditioned Methods of Simultaneous Iteration

It is clear that the application of preconditioning to the iterative methods described in sub-section 2.2.5 will lead to a substantial improvement in the rate of convergence of these methods.

The preconditioned forms of the simultaneous displacement and Richardson's methods are respectively,

\[ \mathbf{y}^{(n+1)} = \mathbf{y}^{(n)} + \alpha(d-B \mathbf{y}^{(n)}) \]  
\[ \mathbf{v}^{(n+1)} = \mathbf{v}^{(n)} + \alpha_n (d-B \mathbf{v}^{(n)}) , \]  

the terms being defined as in sub-section 2.2.6. The iterations proceed in the \( \mathbf{y} \) variable until the specified convergence criteria are met. The final solution is then obtained by one application of the formula

\[ \mathbf{z} = (I-\omega U)^{-1} \mathbf{y} . \]  

If we denote the maximum and minimum eigenvalues of \( B \) by \( \lambda_1(\omega) \) and \( \lambda_N(\omega) \) respectively, then the method (2.92) is convergent for the range of values of \( \alpha \) given by

\[ 0 < \alpha < \frac{2}{\lambda_1(\omega)} \]  

The optimum value of \( \alpha \) is

\[ \alpha = \frac{2}{\lambda_1(\omega)+\lambda_N(\omega)} \]  

and the asymptotic rate of convergence with this choice of \( \alpha \) is

\[ \frac{2\lambda_N(\omega)}{\lambda_1(\omega)} \]  

The optimum \( \alpha \) of method (2.93) are given when the zeros of \( \prod_{i=0}^{N} (1-\alpha_i x) \) coincide with the zeros of the polynomial \( T_{n+1}[(\lambda_1(\omega)+\lambda_N(\omega)-2x)/(\lambda_1(\omega)-\lambda_N(\omega))] \).

The rate of convergence of the method is approximately

\[ 2 \sqrt{\frac{\lambda_N(\omega)}{\lambda_1(\omega)}} \]  

The preconditioned form of the second-order Richardson method is written as

\[ \mathbf{y}^{(n+1)} = \mathbf{y}^{(n)} + \alpha (d-B \mathbf{y}^{(n)}) + \beta (\mathbf{y}^{(n)} - \mathbf{y}^{(n-1)}) , \]  

where
\[ \alpha = \left( \frac{2}{\sqrt{\lambda_1(\omega)} - \sqrt{\lambda_N(\omega)}} \right)^2 \quad \text{and} \quad \beta = \left( \frac{\sqrt{\lambda_1(\omega)} - \sqrt{\lambda_N(\omega)}}{\sqrt{\lambda_1(\omega)} + \sqrt{\lambda_N(\omega)}} \right)^2 \]

the spectral radius is \( \sqrt{\beta} \) and the rate of convergence the same as for the preconditioned Richardson's method.

The preconditioned Chebyshev acceleration of this method has the form,
\[ \chi^{(n+1)} = \chi^{(n)} + \alpha_n (\mathbf{d} \cdot \mathbf{B} \chi^{(n)}) + \beta_n (\chi^{(n)} - \chi^{(n-1)}) \quad (2.100a) \]

where
\[ \alpha_n = \frac{4T_n(\omega(\omega))}{(\sqrt{\lambda_1(\omega)} - \sqrt{\lambda_N(\omega)})_n T_{n+1}(\omega(\omega))} \quad \text{and} \quad \beta_n = \frac{T_{n-1}(\omega(\omega))}{T_{n+1}(\omega(\omega))} \]

and
\[ \omega(\omega) = \frac{\sqrt{\lambda_1(\omega)} + \sqrt{\lambda_N(\omega)}}{\sqrt{\lambda_1(\omega)} - \sqrt{\lambda_N(\omega)}}. \quad (2.100b) \]

The rate of convergence is the same as for the preconditioned Richardson's method.

When \( \alpha \) takes its optimum value in method (2.92), the iteration matrix has two eigenvalues of magnitude \( \omega(\omega) \) (as defined in (2.100b)) and opposite signs. Since all the eigenvalues of \( \mathbf{B} \omega \) are real, the method can be accelerated by Chebyshev polynomials. We can write
\[ \chi^{(n+1)} = \chi^{(n)} + \alpha(\mathbf{d} \cdot \mathbf{B} \chi^{(n)}) \]
followed by
\[ \chi^{(n+1)} = \chi^{(n)} + \alpha_n (\mathbf{d} \cdot \mathbf{B} \chi^{(n)}) + \beta_n (\chi^{(n)} - \chi^{(n-1)}) \]
which combine to give
\[ \chi^{(n+1)} = \chi^{(n)} + \alpha_n (\mathbf{d} \cdot \mathbf{B} \chi^{(n)}) + \beta_n (\chi^{(n)} - \chi^{(n-1)}) \]

where \( \alpha_n = \alpha_n^* \), \( \beta_n = \beta_n^* \). The coefficients \( \alpha_n \) and \( \beta_n \) are as defined in (2.100), except that \( \alpha_0 = 1 \). This is the preconditioned form of the Chebyshev semi-iterative method.

For a matrix derived from a general \( k \)-point finite-difference equation, all of the preconditioned methods described take \( (k-1) \) additions and \( (k-1) \) multiplications per point per iteration more than the corresponding methods without preconditioning.
2.3 COMPUTATIONAL EFFICIENCY OF ITERATIVE METHODS

2.3.1 Relative Execution Times of Floating-Point Instructions

As described in sub-section 2.2.2, it is necessary to consider both the rate of convergence and the computational effort per iteration in evaluating the total work performed by a computer in carrying out an iterative method. In the literature, emphasis is placed on the rate of convergence; computational effort per iteration is rarely presented in detail. In some cases this does not matter. For example, if the coefficient matrix has property A and is consistently ordered, the Gauss-Seidel method converges twice as fast as the Jacobi method using the same number of additions and the same number of multiplications per point per iteration. Hence the Gauss-Seidel method is clearly the more efficient. Comparison is more difficult when the numbers of each type of operation are not the same, as, for example, with SLOR and SPOR. In these cases it is highly desirable to take account of the differing execution times of the various arithmetic operations.

Other factors which contribute to the work per iteration are the accessing of array elements (slower than accessing single named store locations) and testing the absolute difference between elements of successive iterates against some prescribed value. Since the former involves only integer arithmetic, its contribution to the total work is small and can be ignored. It can be argued that it is unnecessary to test for convergence at each iteration since the number of iterations required can be estimated in advance, and testing for convergence started only when this number is approached. However, if the spectral radius of the iteration matrix is not known, the prediction cannot be made. Then it will almost certainly be less work to test for convergence at each iteration than to determine the spectral radius by experiment. While it is often possible to estimate the spectral radius, a small error in the
estimate can cause a large error in the predicted number of iterations. For example, \( p=0.98 \) implies 604 iterations are needed, while \( p=0.99 \) implies 1214 iterations are needed, more than twice the number.

We shall include testing for convergence in our assessments of computational effort. In the solution of linear equations, its contribution is not large enough to significantly affect our conclusions about various iterative methods, but it is included for the sake of completeness. In the solution of non-linear equations in Chapter 4, however, its contribution is significant.

It is impossible to establish a general rule about the relative execution times of instructions as they vary from computer to computer. Furthermore, it is difficult to find average ratios of instruction times over several machines, since manufacturers do not publish comparable data. This is illustrated by Tables 2.6 and 2.7.

**TABLE 2.6**

Floating-point instruction times for Interdata Model 70 processor

All times are given in microseconds

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Minimum</th>
<th>Average</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>23</td>
<td>26</td>
<td>33.5</td>
</tr>
<tr>
<td>Subtract</td>
<td>25</td>
<td>31.5</td>
<td>48.75</td>
</tr>
<tr>
<td>Multiply</td>
<td>54.25</td>
<td>73.5</td>
<td>91.75</td>
</tr>
<tr>
<td>Divide</td>
<td>106.75</td>
<td>109.5</td>
<td>117.25</td>
</tr>
</tbody>
</table>

*(Source - Interdata (1973))*
Relative floating-point instruction times for the ICL 19045* processor at Loughborough University

All times are relative to the unmodified load (LDX) instruction

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
<th>Relative time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fetch instruction</td>
<td>0.56</td>
</tr>
<tr>
<td>2</td>
<td>Modify address</td>
<td>0.11</td>
</tr>
<tr>
<td>3</td>
<td>Load operand into floating-point register</td>
<td>0.79</td>
</tr>
<tr>
<td>4</td>
<td>Execute instruction, Time depends on the function as follows.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Min/U</th>
<th>Max/U</th>
<th>Min/N</th>
<th>Max/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add, subtract</td>
<td>1.36</td>
<td>4.30</td>
<td>1.81</td>
<td>12.67</td>
</tr>
<tr>
<td>Multiply</td>
<td>4.53</td>
<td>4.75</td>
<td>4.98</td>
<td>21.49</td>
</tr>
<tr>
<td>Divide</td>
<td>10.18</td>
<td>11.31</td>
<td>10.41</td>
<td>19.23</td>
</tr>
</tbody>
</table>

Min = minimum calculation (no rounding or alignment)
Max = maximum calculation (with rounding and alignment)
/U = un-normalised operand (no normalising required)
/N = normalised (normalising required)

(Source - Loughborough University Computer Centre (1976))
Comparing the figures in the "average" column of Table 2.6, we see that for this machine, a subtraction operation takes about 1.2 times as long as an addition, a multiplication 2.8 times, a division 4.2 times. The "average" figures given are not the means of the maximum and minimum figures.

In Table 2.7, certain overlapping is possible between the phases of instructions. While phase 4 of one instruction is being carried out, phases 1 and 2 of the next floating-point instruction can be executed. We assume that this always happens and so we need only consider the execution times of phases 3 and 4. We take the average times for phase 4 as the arithmetic means of the four quoted times. This is unlikely to be the true average time, but is the best result which can be found from the available information. Adding the time for phase 3 gives the following average (relative) times.

\[
\begin{array}{|c|c|}
\hline
\text{Instruction} & \text{Average relative time} \\
\hline
\text{Add, subtract} & 5.825 \\
\text{Multiply} & 9.7275 \\
\text{Divide} & 13.5725 \\
\hline
\end{array}
\]

From this table it can be estimated that a multiplication takes about 1.7 times as long as an addition or subtraction, and a division about 2.3 times as long.

It is not sufficient to assess the relative efficiency of algorithms by comparing the running times of programs. The results of the comparisons apply only to the particular language, compiler and computer used. We use typical relative execution times, as given in Table 2.9, to indicate the likely relative efficiency of the algorithms. Clearly, if the actual relative execution times for a particular language, compiler and computer differ substantially from these, the results may not hold.
This will not give a definitive result on the exact relative efficiency of methods for every computer, but will give a better estimate than a simple count of operations.

Convergence tests are carried out by evaluating an arithmetic/logical expression of the form

\[ |a| < b. \]  \hspace{1cm} (2.102)

The simplest way in which this can be evaluated is

(i) if \(a\) is negative, subtract from zero,

(ii) subtract result of stage (i) from \(b\);

(iii) test sign of result of stage (ii).

This involves one or two subtractions plus two tests of sign. Since the latter are fast operations, following Table 2.9 we take a convergence test to have an average relative execution time of 1.5. However, if the computer program is written in a high-level language, the way in which expression (2.102) is actually implemented (in terms of machine code instructions) depends on the compiler. For example, the Algol 68-R compiler for ICL 1900 series computers translates the expression exactly as described above, using five machine-code instructions, while the Fortran compiler XFAT for the same computers uses nineteen machine code instructions and takes about 70% longer to execute. Hence the speed of execution of a program can vary not only between computers but between

---

**TABLE 2.9**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Relative execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>1</td>
</tr>
<tr>
<td>Subtract</td>
<td>1</td>
</tr>
<tr>
<td>Multiply</td>
<td>2.5</td>
</tr>
<tr>
<td>Divide</td>
<td>3.5</td>
</tr>
</tbody>
</table>
programming languages on a particular computer, and this could affect the relative performance of algorithms.

2.3.2 Comparison of Efficiency of Iterative Methods

With the information derived in the previous sub-section, it is possible to make a more detailed comparison than before of the efficiency of point, line two-line and peripheral SOR for the model problem. The work per point per iteration for a matrix derived from a general finite-difference equation depends on the number of mesh points utilised in the equation and for the block methods, the structure of the block. For the model problem, we have

Point SOR: 6 additions
5 multiplications
1 convergence test,

SLOR: 6 additions
5 multiplications
1 convergence test,

S2LOR: 7 additions
6 multiplications
1 convergence test,

SPOR: 8 additions
7 multiplications
1 convergence test.

(2.103)

Using the relative timings derived above, the execution times per point per iteration are as given by Table 2.10.

**TABLE 2.10**

Laplace's equation in the unit square. Work per point per iteration.

<table>
<thead>
<tr>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.0</td>
<td>20.0</td>
<td>23.5</td>
<td>27.0</td>
</tr>
</tbody>
</table>
Taking the number of iterations required for convergence to be inversely proportional to the asymptotic rate of convergence gives the execution times per point relative to point SOR for the converged problem shown in Table 2.11.

**TABLE 2.11**

Laplace's equation in the unit square. Relative work per point per iteration

<table>
<thead>
<tr>
<th>Point</th>
<th>SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.000</td>
<td>0.707</td>
<td>0.588</td>
<td>0.675</td>
<td></td>
</tr>
</tbody>
</table>

With this assumption, S2LOR is the most efficient method, followed by SPOR, SLOR and point SOR. In practice, as shown in Section 2.2, the relationship between the number of iterations and the rate of convergence is not the same for all methods and so Table 2.11 does not indicate the practical relationship between the methods. However, we can obtain an estimate of this by combining the execution times given in Table 2.10 with the experimental results of Tables 2.1 to 2.4 inclusive. This combination of results is given in Table 2.12.

**TABLE 2.12**

Laplace's equation in the unit square. Work per point

<table>
<thead>
<tr>
<th>h^{-1}</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>240</td>
<td>180</td>
<td>164.5</td>
<td>216</td>
</tr>
<tr>
<td>10</td>
<td>500</td>
<td>400</td>
<td>305.5</td>
<td>405</td>
</tr>
<tr>
<td>15</td>
<td>740</td>
<td>560</td>
<td>423</td>
<td>594</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>760</td>
<td>587.5</td>
<td>783</td>
</tr>
<tr>
<td>25</td>
<td>1200</td>
<td>940</td>
<td>705</td>
<td>999</td>
</tr>
<tr>
<td>30</td>
<td>1480</td>
<td>1100</td>
<td>893</td>
<td>1118</td>
</tr>
<tr>
<td>35</td>
<td>1600</td>
<td>1300</td>
<td>1010.5</td>
<td>1377</td>
</tr>
<tr>
<td>40</td>
<td>1800</td>
<td>1500</td>
<td>1151.5</td>
<td>1593</td>
</tr>
</tbody>
</table>
Table 2.13 gives the same information as Table 2.12, but execution times are given relative to point SOR for the particular value of $h^{-1}$.

**TABLE 2.13**

Laplace's equation in the unit square. Relative work per point

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1.000</td>
<td>0.750</td>
<td>0.685</td>
<td>0.900</td>
</tr>
<tr>
<td>10</td>
<td>1.000</td>
<td>0.800</td>
<td>0.611</td>
<td>0.810</td>
</tr>
<tr>
<td>15</td>
<td>1.000</td>
<td>0.757</td>
<td>0.572</td>
<td>0.803</td>
</tr>
<tr>
<td>20</td>
<td>1.000</td>
<td>0.760</td>
<td>0.588</td>
<td>0.783</td>
</tr>
<tr>
<td>25</td>
<td>1.000</td>
<td>0.783</td>
<td>0.588</td>
<td>0.833</td>
</tr>
<tr>
<td>30</td>
<td>1.000</td>
<td>0.743</td>
<td>0.603</td>
<td>0.755</td>
</tr>
<tr>
<td>35</td>
<td>1.000</td>
<td>0.812</td>
<td>0.632</td>
<td>0.861</td>
</tr>
<tr>
<td>40</td>
<td>1.000</td>
<td>0.833</td>
<td>0.640</td>
<td>0.885</td>
</tr>
<tr>
<td>Average over all $h$</td>
<td>1.000</td>
<td>0.780</td>
<td>0.615</td>
<td>0.829</td>
</tr>
</tbody>
</table>

This assessment still gives S2LOR as the most efficient of the four methods, but SPOR is now worse than SLOR, although not by a large margin. No trends are discernable in the figures to suggest that this relationship will be changed for very large values of $h^{-1}$.

It is not possible to derive any general results about the relative efficiencies of preconditioned and un-preconditioned methods, since the improvement in rate of convergence obtained by preconditioning is problem-dependent.

The work per point per iteration for the methods of simultaneous iteration used in this thesis, specified for a matrix with its diagonal elements unity, derived from a general $k$-point finite-difference equation is as follows:
**Simultaneous displacement:**  
- \( k \) additions  
- \( k \) multiplications  
- 1 convergence test  

**Simultaneous displacement with preconditioning:**  
- \( 2k-1 \) additions  
- \( 2k-1 \) multiplications  
- 1 convergence test  

**Second-order Richardson, Chebyshev acceleration, Chebyshev semi-iterative:**  
- \( k+1 \) additions  
- \( k+1 \) multiplications  
- 1 convergence test  

**Second-order Richardson, Chebyshev acceleration, Chebyshev semi-iterative with preconditioning:**  
- \( 2k \) additions  
- \( 2k \) multiplications  
- 1 convergence test  

\[(2.104)\]

Table 2.14 shows the relative execution times per point per iteration for the cases \( k=5,7 \) and 9.

**TABLE 2.14**

Methods of simultaneous iteration. Work per point per iteration

<table>
<thead>
<tr>
<th>( k )</th>
<th>Method</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Simultaneous displacement</td>
<td>19</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>Second-order Richardson, etc.</td>
<td>22.5</td>
<td>36.5</td>
</tr>
<tr>
<td>7</td>
<td>Simultaneous displacement</td>
<td>26</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>Second-order Richardson, etc.</td>
<td>29.5</td>
<td>50.5</td>
</tr>
<tr>
<td>9</td>
<td>Simultaneous displacement</td>
<td>33</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>Second-order Richardson, etc.</td>
<td>36.5</td>
<td>64.5</td>
</tr>
</tbody>
</table>
2.4 EXPERIMENTAL RESULTS FOR PRECONDITIONING WITH MIKHLIN AND SMOLITSKY'S FINITE-DIFFERENCE APPROXIMATION

We solve equation (1.18) on the unit square with $a=b=1$, $e=0.9$, $g(x,y)=0$, using the finite-difference approximation defined by (1.23). We have already shown that the coefficient matrix of this problem is positive definite, and therefore preconditioning theory applies.

The following tables show experimental results for the simultaneous displacement and second-order Richardson method, with and without preconditioning. The maximum and minimum eigenvalues of the coefficient matrix (with and without preconditioning) $\lambda_1$ and $\lambda_N$ respectively are obtained experimentally using the power method. The optimum values of $\omega$ are obtained to within ±0.05 by determining $P(\omega)$ for a range of values of $\omega$ and selecting the one which minimises $P(\omega)$. The optimum parameters $\alpha$ and $\beta$ are calculated from $\lambda_1$ and $\lambda_N$ as described in Section 2.2, and the experimental number of iterations required is found using these values. The theoretical number of iterations is estimated from equation (2.20') where $\varepsilon=5 \times 10^{-6}$.

Table 2.15 shows that preconditioning with the optimum value of $\omega$ for this problem reduces the P-condition number to

$$P(\omega_b) = O(0.6\sqrt{P(0)})$$

(2.105)

which is a very good result, although the multiple of $\sqrt{P(0)}$ increases slightly as $h^{-1}$ increases. Tables 2.16 and 2.17 present experimental results for the simultaneous displacement and second-order Richardson's methods.
Mikhlin and Smolitsky's finite-difference approximation. Eigenvalues of coefficient matrices with and without preconditioning

<table>
<thead>
<tr>
<th>h^{-1}</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>λ₁</td>
<td>λₙ</td>
</tr>
<tr>
<td>5</td>
<td>2.268102</td>
<td>0.134349</td>
</tr>
<tr>
<td>10</td>
<td>2.528710</td>
<td>0.029441</td>
</tr>
<tr>
<td>20</td>
<td>2.597451</td>
<td>0.006360</td>
</tr>
<tr>
<td>30</td>
<td>2.610268</td>
<td>0.002723</td>
</tr>
<tr>
<td>40</td>
<td>2.614820</td>
<td>0.001510</td>
</tr>
</tbody>
</table>
TABLE 2.16
Mikhlin and Smolitsky's finite-difference approximation. Experimental results for the simultaneous displacement method

<table>
<thead>
<tr>
<th>h^{-1}</th>
<th>Without preconditioning</th>
<th>Preconditioning with optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>Iterations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Theory</td>
</tr>
<tr>
<td>5</td>
<td>0.8325</td>
<td>103</td>
</tr>
<tr>
<td>10</td>
<td>0.7818</td>
<td>524</td>
</tr>
<tr>
<td>20</td>
<td>0.7681</td>
<td>2493</td>
</tr>
<tr>
<td>30</td>
<td>0.7654</td>
<td>5851</td>
</tr>
<tr>
<td>40</td>
<td>0.7644</td>
<td>10570</td>
</tr>
</tbody>
</table>
TABLE 2.17
Mikhlin and Smolitsky's finite-difference approximation. Experimental results for the second-order Richardson's method

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Without preconditioning</th>
<th>Preconditioning with optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>$\beta$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.1407</td>
<td>0.3703</td>
</tr>
<tr>
<td>10</td>
<td>1.2887</td>
<td>0.6484</td>
</tr>
<tr>
<td>20</td>
<td>1.3982</td>
<td>0.8203</td>
</tr>
<tr>
<td>30</td>
<td>1.4380</td>
<td>0.8788</td>
</tr>
<tr>
<td>40</td>
<td>1.4588</td>
<td>0.9083</td>
</tr>
</tbody>
</table>
For both the preconditioned simultaneous displacement and preconditioned second-order Richardson methods, the agreement between the predicted and actual number of iterations is satisfactory. For the simultaneous displacement method without preconditioning, however, the theoretical number of iterations over-estimates the actual number required, and the degree of over-estimation rises sharply as \( h^{-1} \) increases (or alternatively, as the spectral radius of the iteration matrix increases). A similar but very much smaller effect can be seen in the results for the second-order Richardson method without preconditioning.

The latter result is surprising, since equation (2.20') from which the theoretical estimates are made, is regarded as giving a lower bound to the number of iterations required.

A possible explanation is as follows. At each iteration, we make the test

\[
\| \tilde{x}^{(n+1)} - x^{(n)} \| = \| e^{(n+1)} - e^{(n)} \| < \epsilon .
\]  

(2.106)

Since

\[
\| e^{(n)} \| = [\rho(M)]^n \| e^{(0)} \| ,
\]

(2.107)
equation (2.106) can be rewritten (approximately) as

\[
\|[\rho(M)]^n[1-\rho(M)]e^{(0)}\| < \epsilon .
\]

(2.108)

Now the left-hand side of this expression is equal to

\[
[\rho(M)]^n[1-\rho(M)]\| e^{(0)} \|
\]

from which it appears that

\[
[\rho(M)]^n[1-\rho(M)] < \epsilon
\]

(2.109)

would give a better estimate of \( n \), the number of iterations required for convergence. In practice this is rarely true because the process takes several iterations to settle down to the state indicated by (2.107) and the effect of the \([1-\rho(M)]\) term in (2.109) is nullified. However, simultaneous displacement is a very simple iterative method and numerical experiments indicate that the condition (2.107) is achieved for very small
values of \( n \). Table 2.18 shows the number of iterations for the simultaneous displacement and second-order Richardson methods, estimated using this modified theory (equation (2.109)).

It can be seen that equation (2.109) gives an excellent estimate of the number of iterations required for convergence for the simultaneous displacement method. The same is not true of the second-order Richardson method, which is reasonable since this is a more complicated iterative process and the effect of the \([1-\rho(M)]\) term is considerably reduced. However, the agreement between theory and experiment does increase a little as \( h^{-1} \) increases (i.e. \( \rho(M) \) increases), which is feasible. The preconditioned versions of these methods are not "simple" and hence the theory developed here does not apply to them.

Table 2.19 gives experimental results for the Chebyshev acceleration of the second-order Richardson method, with and without preconditioning, and the preconditioned Chebyshev semi-iterative method. The Chebyshev semi-iterative method without preconditioning is not applicable, since the eigenvalues \( \lambda_1 \) and \( \lambda_N \) of the coefficient matrix cannot be expressed as \( 1+\mu, 1-\mu \) respectively. (The optimum values of \( \omega \) are used in preconditioning).

We see that the Chebyshev acceleration without preconditioning takes marginally more iterations than the second-order Richardson method without preconditioning, while the reverse is true for the preconditioned forms of the methods. The preconditioned Chebyshev semi-iterative method is very similar to the preconditioned second-order Richardson method.

To compare the overall efficiency of the iterative methods as discussed, it is necessary to consider both the number of iterations required (listed in Tables 2.16, 2.17 and 2.19) and the work per iteration (given in Table 2.14). Table 2.20 indicates the relative work per point for the converged problems, taking simultaneous displacement without preconditioning as the standard.
These results show the very considerable increase in efficiency brought about by the use of the preconditioning technique. As expected, in every case the improvement in the basic method increases as $h^{-1}$ increases.

It was observed that the application of preconditioning improved the accuracy of the solution. In every case, the maximum error in the solution of the preconditioned methods was at least a factor of 10 less than the maximum error in the corresponding methods without preconditioning. This effect has been extensively studied by Hatzopoulos (1974) for direct methods of solving systems of linear equations, but, as far as is known, no comparable studies have been made concerning iterative methods.
### TABLE 2.18

Mikhlin and Smolitsky's finite-difference approximation. Number of iterations required for convergence

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Simultaneous displacement</th>
<th>Second-order Richardson</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Theory [Modified]</td>
<td>Experiment</td>
</tr>
<tr>
<td>5</td>
<td>85</td>
<td>89</td>
</tr>
<tr>
<td>10</td>
<td>363</td>
<td>384</td>
</tr>
<tr>
<td>20</td>
<td>1406</td>
<td>1428</td>
</tr>
<tr>
<td>30</td>
<td>2892</td>
<td>2934</td>
</tr>
<tr>
<td>40</td>
<td>4713</td>
<td>4810</td>
</tr>
<tr>
<td>$h^{-1}$</td>
<td>Chebyshev acceleration</td>
<td>Chebyshev semi-iterative method with preconditioning</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Without preconditioning</td>
<td>With preconditioning</td>
</tr>
<tr>
<td>5</td>
<td>27</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>63</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>124</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td></td>
<td>26</td>
</tr>
<tr>
<td>30</td>
<td>186</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td>37</td>
</tr>
<tr>
<td>40</td>
<td>247</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td>46</td>
</tr>
</tbody>
</table>
TABLE 2.20
Mikhlin and Smolitsky's finite-difference approximation. Work per point

| \( h^{-1} \) | Methods without preconditioning | Methods with preconditioning |
|---|---|---|---|---|---|
| | Simultaneous displacement | Second-order Richardson | Chebyshev acceleration | Simultaneous displacement | Second-order Richardson | Chebyshev acceleration | Chebyshev semi-iterative |
| 5 | 1.000 | 0.382 | 0.344 | 0.325 | 0.240 | 0.218 | 0.240 |
| 10 | 1.000 | 0.189 | 0.186 | 0.159 | 0.091 | 0.081 | 0.091 |
| 20 | 1.000 | 0.093 | 0.099 | 0.100 | 0.040 | 0.033 | 0.035 |
| 30 | 1.000 | 0.069 | 0.072 | 0.078 | 0.025 | 0.021 | 0.024 |
| 40 | 1.000 | 0.056 | 0.058 | 0.066 | 0.018 | 0.015 | 0.019 |
Chapter 3

Successive Over-Relaxation Techniques Applied to Problems in Non-Cartesian Geometries
3.1 INTRODUCTION

This chapter is concerned with the solution of the Poisson equation
\( \nabla^2 \phi = -2 \) in two non-rectangular regions, namely the equilateral triangle and
the parallelogram. It is possible to use a square mesh for these problems,
dealing with irregular mesh points as described in Chapter 2, but it is
much simpler to use a mesh of the same shape as the region and avoid
irregular mesh points altogether.
3.2 Poisson's Equation in an Equilateral Triangle

Let us consider the unit equilateral triangle overlaid with an equilateral-triangular mesh of size $h$. We define the sides of the triangle to be of length $mh=1$, so that there are $\frac{(m-1)(m-2)}{2}$ internal points. Unfortunately, there is no ordering of these points such that scanning over them with the finite-difference equation (1.35) produces a coefficient matrix which has property (A) and is consistently ordered. Hence the theory of point SOR cannot strictly be applied to this problem. However, the coefficient matrix is symmetric and positive definite with positive diagonal elements, and so the SOR method does converge for $0<\omega<2$. What we cannot do is predict the optimum $\omega$ and rate of convergence of the method from the spectral radius of the point Jacobi matrix with any degree of certainty.

In the numerical experiments we use a spiral ordering of points.

If we use this spiral ordering but group all the points on a peripheral circuit together, we obtain a block consistently-ordered matrix which possesses block property (A). Hence we can apply the theory of block SOR. This approach leads to the problem of solving blocks whose coefficient matrix has the form

\[
\begin{pmatrix}
    a_1 & -b_1 & -c_1 \\
    -a_2 & b_2 & -c_2 \\
    & & \ddots \\
    & & & a_j & -b_j & -c_j & -e_j \\
    & & & -a_{j+1} & b_{j+1} & -c_{j+1} \\
    & & & -d_{j+2} & -a_{j+2} & b_{j+2} & -c_{j+2} \\
    & & & & & & \ddots \\
    & & & & & & & a_N & -b_N & -c_N \\
    & & & & & & & -a_{N+1} & b_{N+1} & -c_{N+1} \\
\end{pmatrix}
\]
where \( j = N/3 \), \( N \) being always a multiple of 3.

This matrix can be treated as periodic quindiaagonal and solved by one of the algorithms presented by Benson (1969), pp.111-119. To do so would be inefficient, since two of the five diagonals of elements have only two non-zero elements each. A much better approach is to treat the matrix as periodic tridiagonal with a few extra elements. It can then be solved by a modification of one of the periodic tridiagonal algorithms, mentioned in Chapter 2.

3.2.1 Modified Periodic Tridiagonal Algorithms

Let the matrix \((3.1)\) be \( B \), and the equation to be solved be

\[
Bz = k.
\]

(3.2)

The Gaussian elimination algorithm for this problem transforms equation \((3.2)\) to the form,
\[
\begin{bmatrix}
1 & -g_1 \\
1 & -g_2 \\
1 & -g_j \\
1 & -g_{j+1} \\
1 & -g_{j+2} \\
\end{bmatrix}
\begin{bmatrix}
-1 \\
-1 \\
-1 \\
-1 \\
-1 \\
\end{bmatrix}
\begin{bmatrix}
z_1 \\
z_2 \\
z_{j+1} \\
z_{j+2} \\
z_{j+3} \\
\end{bmatrix} =
\begin{bmatrix}
f_1 \\
f_2 \\
f_{j+1} \\
f_{j+2} \\
f_{j+3} \\
\end{bmatrix}
\]

(3.3a)
For convenience in computing the g's, h's, f's and u's we use the extra vectors $t,D,G,H$ and $F$. To calculate the entries of the matrix in equation (3.3a) we first determine:

\[ D_1 = \frac{1}{b_1} , \quad g_1 = c_1 D_1 , \quad h_1 = a_1 D_1 \]

\[ G_1 = c_N , \quad H_1 = b_N . \]

Then we compute recursively:

\[ D_i = \frac{1}{b_i-a_i g_{i-1}} , \quad g_i = c_i D_i , \quad h_i = a_i h_{i-1} D_i , \]

\[ G_i = g_{i-1} G_{i-1} , \quad H_i = H_{i-1} - G_{i-1} h_{i-1} , \quad i=2,3,\ldots,N-1 \]

with certain exceptions. These are:

when \( i=2 \),

\[ h_i = (a_{N+1} + a_i h_{i-1}) D_i , \]

\[ G_i = c_{N+1} + g_{i-1} G_{i-1} , \]

when \( i=j \) and \( i=2j \),

\[ u_i = e_i D_i , \]

when \( i=j+1 \) and \( i=2j+1 \),

\[ g_i = (c_i + u_{i+1}) D_i , \]

and when \( i=j+2 \) and \( 2j+2 \),

\[ t_i = a_i + g_{i-2} D_i , \]

\[ D_i = \frac{1}{b_i-d_i u_{i-2} - t_i g_{i-1}} , \]

\[ h_i = (t_i h_{i-1} + d_i h_{i-2}) D_i , \]

\[ G_i = u_{i-2} G_{i-2} + g_{i-1} G_{i-1} . \]

Finally we evaluate

\[ D_N = \frac{1}{H_{N-1} - (a_N t_N - h_{N-1}) (g_{N-1} h_{N-1})} . \]

This calculation need only be performed once if the matrix does not change.
The vectors \( \mathbf{f} \) and \( \mathbf{F} \) are computed by forward substitution. Firstly we find

\[
\mathbf{f}_1 = k_1 \mathbf{D}_1, \quad \mathbf{F}_1 = k_N
\]

and then we compute recursively for \( i=2,3,\ldots,N-1 \),

\[
\mathbf{f}_i = (k_i + a_i \mathbf{f}_{i-1}) \mathbf{D}_i, \quad \mathbf{F}_i = \mathbf{F}_{i-1} + G_{i-1} \mathbf{f}_{i-1}
\]

with the exception that when \( i=j+2 \) and \( i=2j+2 \),

\[
\mathbf{f}_i = (k_i + d_i \mathbf{f}_{i-2} + t_i \mathbf{f}_{i-1}) \mathbf{D}_i.
\]

The forward substitution is completed by

\[
\mathbf{F}_N = (\mathbf{F}_{N-1} + (a_N \mathbf{F}_{N-1}) \mathbf{f}_{N-1}) \mathbf{D}_N.
\] (3.3c)

The solution \( \mathbf{z} \) can then be found by back substitution. We set

\[
\mathbf{z}_N = \mathbf{F}_N
\]

and then evaluate recursively for \( i=N-1,N-2,\ldots,1 \)

\[
\mathbf{z}_i = \mathbf{f}_i + g_i \mathbf{z}_{i+1} + h_i \mathbf{z}_N
\]

except that when \( i=j \) and \( i=2j \),

\[
\mathbf{z}_i = \mathbf{f}_i + g_i \mathbf{z}_{i+1} + u_i \mathbf{z}_{i+2} + h_i \mathbf{z}_N.
\] (3.3d)

This algorithm needs in general 4 additions and 5 multiplications per point. The extra arithmetic operations in the 'exception' cases are insignificant except for very small values of \( N \).

Equation (3.2) can also be solved by means of a triangular decomposition algorithm. We factorise the matrix \( \mathbf{B} \) as

\[
\mathbf{B} = \mathbf{L} \mathbf{U},
\] (3.4)

where \( \mathbf{U} \) is an upper triangular matrix with diagonal elements unity, and \( \mathbf{L} \) is a lower triangular matrix. The equation can then be solved using an intermediate vector \( \mathbf{y} \) as follows. We solve

\[
\mathbf{L} \mathbf{y} = \mathbf{k}
\] (3.5a)

by forward substitution, then

\[
\mathbf{U} \mathbf{z} = \mathbf{y}
\] (3.5b)

by back substitution. The forms of the matrices are
\[
L = (3, 6a)
\]

\[
\begin{align*}
\omega_1 & = \beta_1 \\
\omega_2 & = \beta_2 \\
& \vdots \\
\omega_j & = \beta_j \\
& \vdots \\
\omega_{j+2} & = \beta_{j+2} \\
& \vdots \\
\omega_{N} & = \beta_{N} \\
\end{align*}
\]
and

\[
\begin{bmatrix}
1 & -g_1 \\
1 & -g_2 \\
1 & -g_j - u_j \\
1 & -g_{j+1} \\
1 & -g_{j+2} \\
1 & -g_{2j+1} \\
1 & -g_{2j+2} \\
1 & -(g_{N-1} + h_{N-1}) \\
1 & -h_1 \\
1 & -h_2 \\
1 & -h_j \\
1 & -h_{j+1} \\
1 & -h_{j+2} \\
1 & -h_{2j} \\
1 & -h_{2j+1} \\
1 & -h_{2j+2}
\end{bmatrix}
\]

(3.6b)
where \( j = N/3 \). The entries in the \( L \) and \( U \) matrices are calculated as follows.

First we evaluate:

\[
\begin{align*}
\omega_1 &= b_1, \quad \beta_1 = a_2, \quad \gamma_1 = c_N, \\
g_1 &= \frac{c_1}{\omega_1}, \quad h_1 = \frac{a_1}{\omega_1}.
\end{align*}
\]

Then we compute recursively:

\[
\begin{align*}
\omega_i &= b_1 \beta_i-1 g_{i-1} - \beta_i = a_{i+1}, \quad \gamma_i = g_{i-1} \gamma_{i-1}, \\
g_i &= \frac{c_i}{\omega_i}, \quad h_i = \frac{\beta_i-1 h_{i-1}}{\omega_i}, \quad i = 2, 3, \ldots, N-1
\end{align*}
\]

with certain exceptions. These are:

when \( i = 2 \),

\[
h_i = \frac{a_{N+1} + \beta_i-1 h_{i-1}}{\omega_i},
\]

\[
\gamma_i = c_{N+1} + g_i \gamma_1,
\]

when \( i = j \) and \( i = 2j \),

\[
u_i = \frac{e_i}{\omega_i},
\]

\[
\delta_i = d_{i+2},
\]

when \( i = j + 1 \) and \( i = 2j + 1 \),

\[g_i = \frac{c_i + \delta_i-1 u_{i-1}}{\omega_i}\]

\[\beta_i = a_{i+1} + \delta_i-1 g_{i-1},\]

and when \( i = j + 2 \) and \( i = 2j + 2 \)

\[\omega_i = b_1 - \beta_i-1 g_i - \delta_i-2 u_{i-2},\]

\[h_i = \frac{\beta_i-1 h_i + \delta_i-2 h_{i-2}}{\omega_i},\]

\[\gamma_i = u_i-2 \gamma_i-2 + g_i \gamma_i-1.\]

Finally we calculate

\[
\omega_N = b_N - (\beta_{N-1} + \gamma_{N-1})(g_{N-1} + h_{N-1}) - \sum_{i=1}^{N-2} h_i \gamma_i.
\]

(3.6c)
Equation (3.5a) is then solved by forward substitution. The first step is

\[ g_1 = \frac{k_1}{\omega_1} \]

then for \( i=2,3,\ldots,N-1 \)

\[ y_i = \frac{k_i + \beta_{i-1} y_{i-1}}{\omega_i} \]

except that when \( i=j+2 \) and \( i=2j+2 \),

\[ y_i = \frac{k_i + \beta_{i-1} y_{i-1} + \delta_{i-2} y_{i-2}}{\omega_i} \]

Finally we have

\[ y_N = \frac{k_N + \beta_N y_{N-1} + \sum_{i=1}^{N-2} y_i y_i}{\omega_N} \tag{3.6d} \]

Equation (3.5b) can now be solved by the back substitution

\[ z_N = y_N \]

\[ z_i = y_i g_i y_{i+1} + h_i y_N \], \( i=N-1,N-2,\ldots,1 \)

except that when \( i=j \) and \( i=2j \),

\[ z_i = y_i g_i y_{i+1} + u_i y_{i+2} + h_i z_N \tag{3.6e} \]

This algorithm needs an average of 4 additions and 5 multiplications per point, if the inverses of the \( \omega_i \), \( 1 \leq i \leq N \), are computed in (3.6c) and used in (3.6d).

A third method of solving equation (3.2) is a normalised algorithm, but this is only applicable if the matrix \( B \) is symmetric and positive definite. We define the elements of \( B \) by
$B = \begin{bmatrix}
  b_1 & -c_1 \\
  -c_1 & b_2 & -c_2 \\
  & -c_{j-1} & b_j & -c_j & -a_j \\
  & -c_j & b_{j+1} & -c_{j+1} \\
  & -a_j & -c_{j+1} & b_{j+2} & -c_{j+2} \\
  & -c_{2j-2} & b_{2j} & -c_{2j} & -a_{2j} \\
  & -c_{2j} & b_{2j+1} & -c_{2j+1} \\
  & -a_{2j} & -c_{2j+1} & b_{2j+2} & -c_{2j+2} \\
  & -c_{N-2} & b_{N-1} & -c_{N-1} \\
  & -c_{N-1} & b_N \\
  -c_N & -c_{N+1}
\end{bmatrix}$

$N = (3, 7)$
It is well known that a symmetric positive definite matrix has a unique factorisation

$$B = DT^TD$$

where $D$ is a diagonal matrix, $T$ is an upper triangular matrix with ones on the diagonal, and the superscript $T$ denotes the transpose of a matrix. Equation (3.2) can then be written

$$DT^TDz = k,$$  \hspace{1cm} (3.8)

or, putting $h = Dz$, $u = D^{-1}k$,

$$T^Th = u.$$ \hspace{1cm} (3.8')

Let $D$ be $\text{diag}(d_1, d_2, \ldots, d_N)$, and $T$ have the form:
\[
\begin{pmatrix}
1 & -e_1 \\
1 & -e_2 \\
1 & -e_j - e_j \\
1 & -e_{j+1} \\
1 & -e_{j+2} \\
1 & -e_{2j} - e_{2j} \\
1 & -e_{2j+1} \\
1 & -e_{2j+2} \\
1 & -e_{N-1} + e_{N-1} \\
1 & -f_1 \\
-1 & -f_2 \\
-1 & -f_j \\
-1 & -f_{j+1} \\
-1 & -f_{j+2} \\
-1 & -f_{2j} \\
-1 & -f_{2j+1} \\
-1 & -f_{2j+2} \\
0 & 0 \\
0 & 0
\end{pmatrix}
\]

(3.9)
The \(d, e\) and \(g\) elements of the matrices \(D\) and \(T\) are defined by

\[
d_1 = \sqrt{b_1};
\]

\[
d_i = \left\{ b_i - \left( \frac{c_i}{d_i} \right)^2 \right\}^{\frac{1}{2}},
\]

\[
e_{i-1} = \frac{c_i}{d_i} \cdot i = 2, 3, \ldots, N, \]

except that when \(i = j + 2\) and \(i = 2j + 2\),

\[
d_i = \left\{ b_i - \left( \frac{c_i}{d_i} \right) + \frac{a_{i-2} e_{i-2}}{d_{i-2}} - \left( \frac{a_{i-2}}{d_{i-2}} \right)^2 \right\}^{\frac{1}{2}},
\]

\[
e_{i-1} = \frac{c_i}{d_i} d_i + \frac{a_{i-2} e_{i-2}}{d_{i-2} d_i},
\]

and

\[
g_{i-2} = \frac{a_{i-2}}{d_{i-2} d_i}.
\] (3.10a)

In order to compute the value of \(d_N\), we introduce an extra vector \(\gamma_i\) defined by

\[
\gamma_i = f_i d_N, \quad 1 \leq i \leq N - 2
\]

\[
\gamma_{N-1} = (e_{N-1} + f_{N-1}) d_N.
\] (3.10b)

The \(\gamma_i\) are calculated by

\[
\gamma_1 = \frac{c_1}{d_1},
\]

\[
\gamma_2 = \frac{c_1}{d_1} + e_1 \gamma_1,
\]

then for \(i = 2, 3, \ldots, N - 2\),

\[
\gamma_i = e_{i-1} \gamma_{i-1},
\]

except that when \(i = j + 2\) and \(i = 2j + 2\)

\[
\gamma_i = e_{i-1} \gamma_{i-1} + g_{i-2} \gamma_{i-2}.
\]

Finally we have

\[
\gamma_{N-1} = \frac{c_{N-1}}{d_{N-1}} + e_{N-2} \gamma_{N-2}.
\] (3.10c)
Then \( d_N \) is found by

\[
d_N = \left\{ b_N - \sum_{i=1}^{N-1} \gamma_i \right\}
\]  

(3.10d)

and hence we can determine

\[
e_{N-1} = \frac{c_{N-1}}{d_{N-1}d_N} ,
\]

\[
f_1 = \frac{c_N}{d_1 d_N} ,
\]

\[
f_2 = \frac{c_{N+1}}{d_2 d_N} + e_1 f_1 ,
\]

and for \( i=3,4,...,N-1, \)

\[
f_i = e_{i-1} f_{i-1} ,
\]

except that when \( i=j+2 \) and \( i=2j+2 \)

\[
f_i = e_{i-1} f_{i-1} + g_{i-2} f_{i-2} .
\]

(3.10e)

The vector \( u = D^{-1}k \) is computed initially, and the iteration proceeds in the transformed vector \( h \) until convergence is reached, when the transformation

\[
z = D^{-1} h
\]
gives the final solution. Therefore at each iteration it is necessary to solve equation (3.8'). We evaluate an auxiliary vector \( \gamma \) by the forward substitution

\[
T^T \gamma = u
\]

and find the vector \( h \) by backward substitution

\[
Th = \gamma .
\]

In algebraic form the forward substitution is

\[
y_1 = u_1 ;
\]

\[
y_i = u_i + e_{i-1} y_{i-1} , \quad i=2,3,...,N-1,
\]

except for \( i=j+2 \) and \( i=2j+2 \), when we have

\[
y_i = u_i + e_{i-1} y_{i-1} + g_{i-2} y_{i-2}
\]

and finally,

\[
y_N = u_N + e_{N-1} y_{N-1} + \sum_{i=1}^{N-1} f_i y_i .
\]

(3.11a)
The back substitution takes the form

\[ h_N = y_N \]

\[ h_i = y_i + e_i h_{i+1} + f_i h_N, \quad i = N-1, N-2, \ldots, 1, \]

with the exception that when \( i = j \) and \( i = 2j \) we have

\[ h_i = y_i + e_i h_{i+1} + g_i h_{i+2} + f_i h_N. \]  \hspace{1cm} (3.11b)

Neglecting the extra arithmetic in the 'exception' cases, this algorithm needs an average of 4 additions and 4 multiplications per point.

3.2.2 Experimental Results for Point SOR and SPOR

The experimental results presented here are for the equation \( \nabla^2 \phi = -2 \) in the unit equilateral triangle with zero boundary conditions. The initial estimate \( z^{(0)} \) of the solution has all elements unity in every case.

The spectral radii \( \rho(\lambda) \) of the point and peripheral Jacobi iteration matrices are obtained experimentally using the power method. Theoretical values for these spectral radii are not known. The optimum values of \( \omega \) are obtained from equation (2.28). Two estimates of the number of iterations required are made, one assuming a single eigenvalue whose magnitude is the spectral radius (equation (2.20')) and the other assuming two eigenvalues of this magnitude (equation (2.21)), with \( \epsilon = 5 \times 10^{-6} \). These estimates are labelled Theory 1 and Theory 2 in the following tables.


<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th></th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td>30</td>
<td>0.985433</td>
<td>36</td>
<td>48</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>45</td>
<td>0.993515</td>
<td>53</td>
<td>74</td>
<td>68</td>
</tr>
<tr>
<td>60</td>
<td>0.996352</td>
<td>71</td>
<td>100</td>
<td>90</td>
</tr>
<tr>
<td>90</td>
<td>0.998386</td>
<td>107</td>
<td>153</td>
<td>131</td>
</tr>
</tbody>
</table>

Poisson's equation in the unit equilateral triangle, Point SOR
TABLE 3.2
Poisson's equation in the unit equilateral triangle. Peripheral SOR

| h^{-1} | \rho(B) | \begin{tabular}{c|c|c|}
|   | Iterations | \hline
<table>
<thead>
<tr>
<th></th>
<th>Theory</th>
<th>Experiment</th>
<th>Theory</th>
<th>Experiment</th>
<th>Optimum \omega</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>0.977275</td>
<td>28</td>
<td>38</td>
<td>31</td>
<td>1.65</td>
</tr>
<tr>
<td>45</td>
<td>0.989966</td>
<td>43</td>
<td>59</td>
<td>47</td>
<td>1.75</td>
</tr>
<tr>
<td>60</td>
<td>0.994343</td>
<td>57</td>
<td>79</td>
<td>62</td>
<td>1.81</td>
</tr>
<tr>
<td>90</td>
<td>0.997433</td>
<td>85</td>
<td>120</td>
<td>91</td>
<td>1.87</td>
</tr>
</tbody>
</table>
In the SPOR results table, the agreement between theoretical and experimental values of \( w \) is excellent as expected. The number of iterations required agrees closely with the estimates assuming a single eigenvalue whose magnitude is the spectral radius. This result is similar to that for Laplace's equation in the unit square (Chapter 2) and is further evidence that the lack of influence of the second eigenvalue whose magnitude is the spectral radius is a feature of the peripheral form of SOR.

Despite the fact that the SOR theory does not strictly apply, the estimates of \( w \) in the point method agree well with the experimental results. The number of iterations lies midway between the two theoretical values, a result also seen in the unit square problem. Hence for practical purposes it may be assumed that the theory of SOR is applicable.

We see that SPOR converges in about two-thirds the number of iterations of SOR. To obtain a full picture of the relationship between the methods, it is necessary to consider the amount of work for each iteration. As indicated in the previous chapter, this consists of

8 additions,
7 multiplications,
1 convergence test for point SOR

and

10 additions,
9 multiplications,
1 convergence test for SPOR.

Using the weighting scheme developed earlier, we obtain the following results for execution times per point for the converged problem.
TABLE 3.3

Poisson's equation in the unit equilateral triangle. Work per point

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Point SOR</th>
<th>SPOR</th>
<th>Ratio $\frac{SPOR}{Point \ SOR}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1242</td>
<td>1054</td>
<td>0.849</td>
</tr>
<tr>
<td>45</td>
<td>1836</td>
<td>1598</td>
<td>0.870</td>
</tr>
<tr>
<td>60</td>
<td>2430</td>
<td>2108</td>
<td>0.867</td>
</tr>
<tr>
<td>90</td>
<td>3537</td>
<td>3094</td>
<td>0.875</td>
</tr>
</tbody>
</table>

On this set of results, it appears that SPOR takes about 0.87 as much computer time as point SOR, which is a worthwhile saving.

3.2.3 Variations of and Extensions to the Basic SPOR Method

Within each peripheral, the mesh points can be numbered alternately, as in Figure 3.1.
This leads to block matrices whose structure is fundamentally quind diagonal, with four non-zero elements outside the five diagonals. The work required to solve these blocks is the same as for the serial ordering, and the method does not offer any particular advantages over the original one.

It is possible to solve blocks alternately, the odd-numbered ones in sequence followed by the even-numbered ones in sequence (or vice versa). In theory this makes no difference to the rate of convergence of the method. In practice there are minor discrepancies in the number of iterations required for non-optimal values of $\omega$.

The question arises of whether a 2-peripheral block SOR would be more efficient than a 1-peripheral version. Certainly we would expect its rate of convergence to be larger. Unfortunately it is impossible to devise an ordering of mesh points which gives a compactly banded block matrix. Although a matrix can be constructed which has a fundamentally periodic quindiagonal form, the effect of the vertices of the triangle is to scatter a few non-zero elements widely across the matrix. Any algorithm to solve the matrix must therefore be complicated and of low efficiency so that 2-peripheral SOR is unlikely to be worthwhile.

3.2.4 A Note on Computer Implementation of Problems in a Triangular Region

We consider the problem of designing a suitable data structure for the vector of approximate solutions. The ideal is a triangular array, but high-level programming languages allow only rectangular arrays.

One possible solution is to set up a square array and use one (triangular) half of it. This is very wasteful of memory, unless it is necessary to store a value of the right-hand side of the matrix equation for each mesh point, in which case the other half of the array can be utilised.

Memory can be solved by using a one-dimensional array and accessing
elements via a function which maps two co-ordinates of a conceptual triangular array to a single co-ordinate of the actual array. This solution, however, adds enormously to the program's execution time.

A data structure which minimises both memory and processing requirements can be declared in Algol 68. We set up a one-dimensional array of addresses (references), the elements of which point to one-dimensional arrays (of different sizes) of real numbers. Hence we effectively have a triangular array. To declare such a structure (x, say) with n elements along each edge, we write

\[ [1:n]\text{REF[]REAL } x; \]

which sets up the vector to hold the addresses, then

\[
\text{INT } i:=1; \\
\text{label:} \\
\quad x[i]:=\text{LOC}[1:i]\text{REAL}; \\
\quad i \text{ PLUS 1;} \\
\quad \text{IF } i\leq n \text{ THEN GOTO label} \\
\text{FI;}
\]

which sets up the vectors of real numbers and their addresses. Elements of the structure are accessed thus:

\[ x[i][j] \]

and the neighbouring mesh points have subscripts

\[
i\pm 1,j, \\
i,j\pm 1, \\
i+1,j-1, \\
i-1,j+1.
\]
Let us consider a parallelogram overlaid with a skew mesh of length \( h \) in the \( u \)-direction and length \( k \) in the \( v \)-direction, as in Figure 3.2.

We wish to scan over the mesh points with equation (1.42) to produce a coefficient matrix which if possible has property (A) and consistent ordering (or the block equivalents of these properties) so that the SOR theory holds.

Unfortunately, no ordering of the mesh points exists such that point SOR theory applies. However, the natural ordering of points produces a coefficient matrix which is symmetric and positive definite with positive diagonal elements, so that the point SOR method does converge for \( 0 < \omega < 2 \).
There are several orderings of the mesh points which lead to block tridiagonal coefficient matrices, which possess block property (A) and are block consistently ordered. We shall consider the three SOR methods illustrated for the square region in Chapter 2, namely the line, two-line and peripheral SOR methods. Let the coefficient matrix have the form

\[
\begin{bmatrix}
B_1 & C_1 \\
A_2 & B_2 & C_2 \\
& A_{r-1} & B_{r-1} & C_{r-1} \\
& & A_r & B_r \\
\end{bmatrix}
\]

(3.13a)

where the terms are defined as follows. For line SOR we take the lines parallel to the u-axis so that \(r=n\) and the sub-matrices are defined by:

\[
\begin{bmatrix}
2(1+\lambda^2) & -1 \\
-1 & 2(1+\lambda^2) & -1 & 0 \\
& -1 & 2(1+\lambda^2) & -1 \\
& & -1 & 2(1+\lambda^2) \\
\end{bmatrix}_{m\times m}
\]

(3.13b)

and

\[
\begin{bmatrix}
-\lambda^2 & \frac{\lambda \cos a}{2} \\
-\frac{\lambda \cos a}{2} & -\lambda^2 & \frac{\lambda \cos a}{2} & 0 \\
& -\frac{\lambda \cos a}{2} & -\lambda^2 & \frac{\lambda \cos a}{2} \\
& & -\frac{\lambda \cos a}{2} & -\lambda^2 & \frac{\lambda \cos a}{2} \\
\end{bmatrix}_{m\times m}
\]

For two-line SOR, we again take the lines to be parallel to the u-axis so that \(r=n/2\) (assuming \(n\) to be even) and the sub-matrices of (3.13a) are then defined by
\[
B_i = \begin{bmatrix}
2(1+\lambda^2) & -\lambda^2 & -1 & \frac{\lambda \cos a}{2} \\
-\lambda^2 & 2(1+\lambda^2) & -\frac{\lambda \cos a}{2} & -1 & 0 \\
-1 & -\frac{\lambda \cos a}{2} & 2(1+\lambda^2) & -\lambda^2 & -1 & \frac{\lambda \cos a}{2} \\
\frac{\lambda \cos a}{2} & -1 & -\lambda^2 & 2(1+\lambda^2) & -\frac{\lambda \cos a}{2} & -1 & 0 \\
0 & -1 & -\frac{\lambda \cos a}{2} & 2(1+\lambda^2) & -\lambda^2 & -1 & \frac{\lambda \cos a}{2} \\
\frac{\lambda \cos a}{2} & -1 & -\lambda^2 & 2(1+\lambda^2) & -\frac{\lambda \cos a}{2} & -1 \\
0 & -1 & -\frac{\lambda \cos a}{2} & 2(1+\lambda^2) & -\lambda^2 & -1 & \frac{\lambda \cos a}{2} \\
0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 \\
\end{bmatrix}_{2m \times 2m}
\]
and

\[
\begin{bmatrix}
0 & 0 & 0 \\
-\lambda^2 & 0 & \frac{\lambda \cos \alpha}{2} \\
0 & 0 & 0 \\
-\frac{\lambda \cos \alpha}{2} & 0 & -\lambda^2 & 0 & \frac{\lambda \cos \alpha}{2} \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix},
\]

\[A_{i+1} = C_i^T = \begin{bmatrix}
\frac{-\lambda \cos \alpha}{2} & 0 & -\lambda^2 & 0 & \frac{\lambda \cos \alpha}{2} \\
0 & \frac{-\lambda \cos \alpha}{2} & 0 & -\lambda^2 & 0 & \frac{\lambda \cos \alpha}{2} \\
0 & 0 & 0 & 0 & 0 \\
\frac{-\lambda \cos \alpha}{2} & 0 & -\lambda^2 & 0 & \frac{\lambda \cos \alpha}{2} \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}_{2m \times 2m}
\]

, \text{ for } \xi \geq -1. \quad (3.13c)
In the peripheral SOR method, \( r = \min \left( \frac{m}{2}, \frac{n}{2} \right) \). The sub-matrices \( B_i \) now take the form:

\[
\begin{bmatrix}
b_1 & -c_1 \\
-a_2 & b_2 & -c_2 \\
& a_j & b_j & -c_j & -e_j \\
& -a_{j+1} & b_{j+1} & -c_{j+1} \\
& -d_{j+2} & -a_{j+2} & b_{j+2} & -c_{j+2} \\
& & & a_{2j} & b_{2j} & -c_{2j} & -e_{2j} \\
& & & -a_{2j+1} & b_{2j+1} & -c_{2j+1} \\
& & & -d_{2j+2} & -a_{2j+2} & b_{2j+2} & -c_{2j+2} \\
& & & & a_{3j} & b_{3j} & -c_{3j} & -e_{3j} \\
& & & & -a_{3j+1} & b_{3j+1} & -c_{3j+1} \\
& & & & -d_{3j+2} & -a_{3j+2} & b_{3j+2} & -c_{3j+2} \\
& & & & & & a_{N-1} & b_{N-1} & -c_{N-1} \\
& & & & & & -a_N & b_N & -c_{N+1}
\end{bmatrix}
\]

(3.13d)
where \( j = \frac{N}{4} \), \( N \) being always a multiple of 4. The non-zero entries in this matrix are defined as follows.

The \( a_i \) and \( c_i \) elements have the values

\[
 a_i = c_i = \lambda^2, \quad a_i = c_{i-1} = 1, \quad 1 \leq i \leq j \quad \text{and} \quad 2j+1 \leq i \leq 3j,
\]

\[
 a_i = c_{i-1} = \lambda^2, \quad j+1 \leq i \leq 2j \quad \text{and} \quad 3j+1 \leq i \leq N,
\]

\[
 a_{N+1} = c_{N+1} = \frac{\lambda}{2 \cos \alpha}. \tag{3.13e}
\]

The \( d \) and \( e \) elements are defined by

\[
 d_{j+2} = e_j = d_{3j+2} = e_{3j} = -\frac{\lambda \cos \alpha}{2},
\]

\[
 d_{2j+2} = e_{2j} = \frac{\lambda \cos \alpha}{2}. \tag{3.13f}
\]

From (3.13e) and (3.13f), we see that the matrix is symmetric. Finally, the \( b \) terms are given by

\[
 b_i = 2(1+\lambda^2), \quad 1 \leq i \leq N. \tag{3.13g}
\]

For the off-diagonal sub-matrices

\[
 A_{i+1} = C_i^T, \quad 1 \leq i \leq N-1
\]

and their structure, illustrated here for a \((16 \times 8)\) sub-matrix is
\[
\Lambda = \begin{pmatrix}
\frac{\lambda \cos a}{2} & -\lambda^2 & -\frac{\lambda \cos a}{2} & -\frac{\lambda \cos a}{2} \\
\frac{\lambda \cos a}{2} & -\lambda^2 & -\frac{\lambda \cos a}{2} & -\frac{\lambda \cos a}{2} \\
\frac{\lambda \cos a}{2} & -\lambda^2 & -\frac{\lambda \cos a}{2} & -\frac{\lambda \cos a}{2} \\
\frac{\lambda \cos a}{2} & -\lambda^2 & -\frac{\lambda \cos a}{2} & -\frac{\lambda \cos a}{2} \\
0 & \frac{\lambda \cos a}{2} & -\lambda^2 & -\frac{\lambda \cos a}{2} \\
0 & 0 & \frac{\lambda \cos a}{2} & -\lambda^2 \\
0 & 0 & 0 & \frac{\lambda \cos a}{2} \\
-\frac{\lambda \cos a}{2} & -1 & \frac{\lambda \cos a}{2} & -1
\end{pmatrix}
\]
(3.13h)
The block SOR process for a coefficient matrix of the form (3.13a) is as described by equation (2.35). The key difference between problems in a rectangle formulated using a five-point finite-difference equation (as presented in Chapter 2) and problems in a parallelogram formulated using a nine-point finite-difference equation is the structure of the matrices $B_i$. In each of the block SOR methods, we require to solve sets of equations of the form

$$B_i z_i = k_i , \quad 1 \leq i \leq r .$$  \hspace{1cm} (3.14)

As shown above, the line SOR method produces matrices $B_i$ which are tridiagonal. In this case, equation (3.14) can be solved by the normalised triangular decomposition algorithm mentioned in Chapter 2, which requires 2 additions and 2 multiplications per point. The $B_i$ matrices of the two-line SOR method are seven-diagonal, with a maximum of six non-zero elements on each line. An algorithm for the solution of (3.14) in this case is presented below. The peripheral SOR method produces $B_i$ matrices which are fundamentally periodic tridiagonal with a few extra non-zero elements. They differ from the $B_i$ matrices of the peripheral SOR method in an equilateral triangle (see equation (3.1)) in having three non-zero elements in the second super- and second sub-diagonals instead of two. Algorithms for the solution of equation (3.14) can be simply derived from the algorithms in section 3.2 as follows.

Let us consider the various vectors and sets of (non-zero) matrix entries computed in the latter algorithms. The elements of any particular vector or set of matrix entries are defined similarly for the range of subscripts $3 \leq i \leq N - 2$, with some exceptions. The exceptions are due to the presence of the two non-zero elements in the second super- and second sub-diagonals, and occur at pairs of values of $i$ either at $i = \frac{N}{3}$ and $i = \frac{2N}{3}$, or at $i = \frac{N}{3} + 1$ and $i = \frac{2N}{3} + 1$, or at $i = \frac{N}{3} + 2$ and $i = \frac{2N}{3} + 2$. 

depending on the vector or set of matrix entries. In the parallelogram problem algorithms, there are three non-zero elements in the second super- and second sub-diagonals, so that the exceptions occur at a triple of values of $i$,
either at $i = \frac{N}{4}$, $i = \frac{N}{2}$ and $i = \frac{3N}{4}$,
or at $i = \frac{N}{4} + 1$, $i = \frac{N}{2} + 1$ and $i = \frac{3N}{4} + 1$,
or at $i = \frac{N}{4} + 2$, $i = \frac{N}{2} + 2$ and $i = \frac{3N}{4} + 2$
respectively. The normalised algorithm is applicable since the matrices $B_i$ are symmetric, and can be shown to be irreducibly diagonally dominant and hence positive definite. This algorithm requires 4 additions and 4 multiplications per point.

3.3.1 A Modified Seven-Diagonal Algorithm

It is required to solve the matrix equation (3.14) where $B_i$ has the form

\[
\begin{bmatrix}
-1 & -e_1 & -f_1 & -g_1 \\
-c_2 & d_2 & -e_2 & -f_2 \\
-b_3 & -c_3 & d_3 & -e_3 & -f_3 & -g_3 \\
-a_4 & -b_4 & -c_4 & d_4 & -e_4 & -f_4 & 0 \\
-b_5 & -c_5 & d_5 & -e_5 & -f_5 & -g_5 \\
-a_6 & -b_6 & -c_6 & d_6 & -e_6 & -f_6 & 0 \\
&&&&&&&&
\end{bmatrix}
\]  

\[(3.15a)\]
Since the matrix is derived from two lines of points, \( N \) is always even. (To avoid confusion, we omit the subscripts of equation (3.14) from here onwards).

A Gaussian elimination process can be used to transform equation (3.14) with this structure of \( B \) to

\[
\begin{bmatrix}
1 & -\alpha_1 & -\beta_1 & \gamma_1 \\
1 & -\alpha_2 & -\beta_2 & \\
1 & -\alpha_3 & -\beta_3 & -\gamma_3 \\
1 & -\alpha_4 & -\beta_4 & \\
1 & -\alpha_5 & -\beta_5 & -\gamma_5 \\
1 & -\alpha_6 & -\beta_6 & \\
0 & & & 0 \\
0 & & & 0 \\
1 & -\alpha_{N-3} & -\beta_{N-3} & -\gamma_{N-3} \\
1 & -\alpha_{N-2} & -\beta_{N-2} & \\
1 & -\alpha_{N-1} & -\beta_{N-1} & \\
1 & -\alpha_N & -\beta_N & \\
1 & & & \\
0 & & & \\
\end{bmatrix}
\begin{bmatrix}
z_1 \\
z_2 \\
z_3 \\
z_4 \\
z_5 \\
z_6 \\
z_N-3 \\
z_N-2 \\
z_N-1 \\
z_N \\
\end{bmatrix}
= 
\begin{bmatrix}
h_1 \\
h_2 \\
h_3 \\
h_4 \\
h_5 \\
h_6 \\
h_{N-3} \\
h_{N-2} \\
h_{N-1} \\
h_N \\
\end{bmatrix}
\tag{3.15b}
\]

We define the non-zero entries in this matrix, using the extra vectors \( \omega, \delta \) and \( \epsilon \), as follows:

Initially we set

\[
\alpha_{-1} = \alpha_0 = 0, \quad \beta_{-1} = \beta_0 = 0, \quad \gamma_{-1} = \gamma_0 = 0
\]

and

\[
\omega_1 = \frac{1}{d_1}, \quad \alpha_1 = \epsilon_1 \omega_1, \quad \beta_1 = \delta_1 \omega_1, \quad \gamma_1 = \gamma_1 \omega_1.
\]

The definitions of the entries in subsequent rows depend on whether the number of the row is odd or even. For even-numbered rows we have

\[
\delta_i = b_i + a_i \alpha_{i-3},
\]

\[
\epsilon_i = c_i + \delta_i \alpha_{i-2} + a_i \beta_{i-3},
\]

\[
\omega_i = \frac{1}{d_i - \epsilon_i a_{i-1} - \delta_i \beta_{i-2} - a_i \gamma_{i-3}}.
\]
\[ a_i = (e_i + \delta_i \beta_{i-1}) \omega_i \]

and

\[ \beta_i = (f_i + \delta_i \gamma_{i-1}) \omega_i, \quad i=2,4,\ldots,N \]

except that

\[ \alpha_N = \beta_N = 0. \]

The entries in the odd-numbered rows are defined by

\[ \delta_i = b_i, \]

\[ \epsilon_i = c_i + \delta_i \epsilon_{i-2}, \]

\[ \omega_i = \frac{1}{d_i - \epsilon_i \delta_{i-1} - \delta_i \beta_{i-2}}, \]

\[ \alpha_i = (e_i + \epsilon_i \beta_{i-1} + \delta_i \gamma_{i-2}) \omega_i, \]

\[ \beta_i = f_i \omega_i, \]

and

\[ \gamma_i = g_i \omega_i, \quad i=3,5,\ldots,N-1 \]

except that

\[ \beta_{N-1} = \gamma_{N-1} = 0. \quad (3.15c) \]

(The entries must be evaluated strictly in order of row number).

The right-hand side vector \( h \) is computed by forward substitution. We evaluate

\[ h_1 = k_1 \omega_1, \]

\[ h_2 = (k_2 + \epsilon_2 h_1) \omega_2, \]

then \( h_3, h_4, \ldots, h_N \), using for odd-numbered rows the equation

\[ h_i = (k_i + \epsilon_i h_{i-1} + \delta_i h_{i-2}) \omega_i, \quad i=3,5,\ldots,N-1 \]

and for even-numbered rows the equation

\[ h_i = (k_i + \epsilon_i h_{i-1} + \delta_i h_{i-2} + \alpha_i h_{i-3}) \omega_i, \quad i=4,6,\ldots,N. \]

The solution \( z \) can then be found by back substitution. We calculate

\[ z_N = h_N, \]

\[ z_{N-1} = h_{N-1} + \alpha_{N-1} z_N, \]

then \( z_{N-2}, z_{N-3}, \ldots, 1 \) using for even-numbered rows the equation

\[ z_i = h_i + \alpha_i z_{i+1} + \beta_i z_{i+2}, \quad i=N-2, N-4, \ldots, 2 \]
and for odd-numbered rows the equation

\[ z_i = h_{i+1} z_{i+1} + h_{i+2} z_{i+2} + z_{i+3}, \quad i = N-3, N-5, \ldots, 1. \] (3.15d)

The actual solution process of the matrix equation (3.14), represented by the equations (3.15d), requires 5 additions and 6 multiplications per point.

The question arises as to whether a normalised triangular decomposition algorithm can be used to solve the equation. If such an algorithm exists, it will require 5 additions and 5 multiplications per point. From equation (3.13c), the matrix B is clearly symmetric and can be shown to be irreducible. The matrix can be proved to be positive definite by showing that it is diagonally dominant and that strict diagonal dominance holds for at least one row (Theorem 1.6). We require,

\[ 2(1+\lambda^2) \geq 1 + \lambda^2 + \frac{\lambda \cos \alpha}{2} \] (3.16)

and

\[ 2(1+\lambda^2) \geq 2 + \lambda^2 + \lambda \cos \alpha. \] (3.17)

It can be shown that strict inequality always holds in equation (3.16), while (3.17) is true provided

\[ \lambda > \cos \alpha, \] (3.18)

which is certainly true for \( \lambda > 1 \). Hence if \( \lambda \) is at least unity, the normalised algorithm can be used.

### 3.3.2 Experimental Results for Point and Block SOR

The experimental results presented here are for the equation \( \nabla^2 \phi = -2 \) in the unit rhombus with zero boundary conditions. We choose \( n=m \) and therefore \( h=k \), i.e., the mesh lengths and numbers of points are the same in both directions and \( \lambda = 1 \). The initial estimate \( z^{(0)} \) of the solution has all elements unity in every case.

The spectral radii \( \rho(B) \) of the point and block Jacobi matrices are obtained experimentally using the power method, and the optimum values
of $\omega$ computed from them using equation (2.28). Two estimates are made of
the number of iterations required, one assuming a single eigenvalue whose
magnitude is the spectral radius (equation (2.20')) and the other assuming
two eigenvalues of this magnitude (equation (2.21)) with $\epsilon = 5 \times 10^{-6}$. These
are labelled Theory 1 and Theory 2.
TABLE 3.4

Poisson's equation in a unit rhombus. Point SOR.a = \frac{\pi}{2}

<table>
<thead>
<tr>
<th>h^{-1}</th>
<th>\rho(B)</th>
<th>Iterations</th>
<th>Optimum \omega</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0.809017</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>0.951057</td>
<td>19</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>0.987688</td>
<td>39</td>
<td>53</td>
</tr>
<tr>
<td>40</td>
<td>0.996925</td>
<td>78</td>
<td>109</td>
</tr>
</tbody>
</table>
TABLE 3.5

Poisson's equation in a unit rhombus. Line SOR, $\alpha = \frac{\pi}{2}$

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>$\text{Iterations}$</th>
<th>$\text{Optimum } \omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td>5</td>
<td>0.679285</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>0.906680</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>0.975676</td>
<td>27</td>
<td>37</td>
</tr>
<tr>
<td>40</td>
<td>0.993885</td>
<td>55</td>
<td>76</td>
</tr>
</tbody>
</table>
TABLE 3.6

Poisson's equation in a unit rhombus. Two-ine SOR, $\alpha = \frac{\pi}{2}$

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td>5</td>
<td>0.509646</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>0.829983</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>20</td>
<td>0.952521</td>
<td>19</td>
<td>26</td>
</tr>
<tr>
<td>40</td>
<td>0.987783</td>
<td>39</td>
<td>53</td>
</tr>
</tbody>
</table>
TABLE 3.7

Poisson's equation in the unit rhombus. Peripheral SOR, $\alpha = \frac{\pi}{2}$

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0.632456</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>0.907015</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>0.975682</td>
<td>27</td>
<td>37</td>
</tr>
<tr>
<td>40</td>
<td>0.993849</td>
<td>55</td>
<td>76</td>
</tr>
</tbody>
</table>
TABLE 3.8

Poisson's equation in the unit rhombus. Point SOR, $\alpha = \frac{\pi}{3}$

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory 1</td>
<td>Theory 2</td>
</tr>
<tr>
<td>5</td>
<td>0.814329</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>0.953283</td>
<td>20</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>0.988324</td>
<td>40</td>
<td>54</td>
</tr>
<tr>
<td>40</td>
<td>0.997082</td>
<td>80</td>
<td>112</td>
</tr>
</tbody>
</table>
TABLE 3.9
Poisson's equation in the unit rhombus. Line SOR, $\omega = \frac{\pi}{3}$

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td>5</td>
<td>0.689970</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>0.911151</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>0.976949</td>
<td>28</td>
<td>38</td>
</tr>
<tr>
<td>40</td>
<td>0.994199</td>
<td>57</td>
<td>78</td>
</tr>
</tbody>
</table>
TABLE 3.10

Poisson's equation in the unit rhombus. Two-line SOR, $\alpha = \frac{\pi}{3}$

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th></th>
<th>Optimum $\omega$</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
<td>Theory</td>
</tr>
<tr>
<td>5</td>
<td>$0.522861$</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>1.08</td>
</tr>
<tr>
<td>10</td>
<td>$0.837461$</td>
<td>10</td>
<td>14</td>
<td>14</td>
<td>1.29</td>
</tr>
<tr>
<td>20</td>
<td>$0.954951$</td>
<td>20</td>
<td>27</td>
<td>26</td>
<td>1.54</td>
</tr>
<tr>
<td>40</td>
<td>$0.988433$</td>
<td>40</td>
<td>54</td>
<td>50</td>
<td>1.74</td>
</tr>
<tr>
<td>$h^{-1}$</td>
<td>$\rho(B)$</td>
<td>Iterations</td>
<td>Optimum $\omega$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>-------------</td>
<td>-----------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Theory 1</td>
<td>Theory 2</td>
<td>Experiment</td>
<td>Theory</td>
</tr>
<tr>
<td>5</td>
<td>0.646964</td>
<td>6</td>
<td>9</td>
<td>8</td>
<td>1.13</td>
</tr>
<tr>
<td>10</td>
<td>0.912462</td>
<td>14</td>
<td>19</td>
<td>16</td>
<td>1.42</td>
</tr>
<tr>
<td>20</td>
<td>0.977103</td>
<td>28</td>
<td>38</td>
<td>31</td>
<td>1.65</td>
</tr>
<tr>
<td>40</td>
<td>0.994194</td>
<td>57</td>
<td>78</td>
<td>61</td>
<td>1.81</td>
</tr>
</tbody>
</table>

**TABLE 3.11**

Poisson's equation in a unit rhombus. Peripheral SOR, $\alpha = \frac{\pi}{3}$
TABLE 3.12

Poisson's equation in the unit rhombus. Point \( \text{SOR.} \alpha = \frac{n}{12} \)

<table>
<thead>
<tr>
<th>( h^{-1} )</th>
<th>( \rho(B) )</th>
<th>Iterations</th>
<th>Optimum ( \omega )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td>5</td>
<td>0.830362</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>10</td>
<td>0.961510</td>
<td>22</td>
<td>29</td>
</tr>
<tr>
<td>20</td>
<td>0.991080</td>
<td>46</td>
<td>62</td>
</tr>
<tr>
<td>40</td>
<td>0.997861</td>
<td>93</td>
<td>132</td>
</tr>
</tbody>
</table>
**TABLE 3.13**

Poisson's equation in the unit rhombus. Line SOR. \( \alpha = \frac{\pi}{12} \)

<table>
<thead>
<tr>
<th>( h^{-1} )</th>
<th>( \rho(B) )</th>
<th>Iterations</th>
<th>Optimum ( \omega )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Theory</strong></td>
<td><strong>Experiment</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>0.722579</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>0.928050</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>20</td>
<td>0.982544</td>
<td>32</td>
<td>44</td>
</tr>
<tr>
<td>40</td>
<td>0.995734</td>
<td>66</td>
<td>92</td>
</tr>
</tbody>
</table>
TABLE 3.14

Poisson's equation in the unit rhombus. Two-line SOR, $\alpha = \frac{\pi}{12}$

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0.563292</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>0.866239</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>20</td>
<td>0.965693</td>
<td>23</td>
<td>31</td>
</tr>
<tr>
<td>40</td>
<td>0.991539</td>
<td>47</td>
<td>64</td>
</tr>
</tbody>
</table>
TABLE 3.15

Poisson's equation in the unit rhombus. Peripheral SOR, $\alpha = \frac{\pi}{12}$

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
</tr>
<tr>
<td>5</td>
<td>0.686874</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>0.932028</td>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>20</td>
<td>0.983278</td>
<td>33</td>
<td>45</td>
</tr>
<tr>
<td>40</td>
<td>0.995846</td>
<td>67</td>
<td>93</td>
</tr>
</tbody>
</table>
In the preceding tables, the theoretical and experimental optimum values of $\omega$ agree closely, as expected, for the block SOR methods.

Agreement is also good for the point SOR method when $\alpha = \frac{\pi}{2}$ (Table 3.4); however, for this value of $\alpha$, the problem reduces to our model problem of Chapter 2, where the point SOR theory holds. The theory does not hold for other values of $\alpha$, as can be seen from Table 3.8, where $\alpha = \frac{\pi}{3}$, and the theory slightly over-estimates the optimum $\omega$ instead of slightly under-estimating it (the latter being usual when the SOR theory holds). Clearer evidence is provided by Table 3.12, where $\alpha = \frac{\pi}{12}$, and the theory grossly over-estimates the optimum $\omega$. The results for $\alpha = \frac{\pi}{2}$ are identical (within the limits of experimental error) to the results for Laplace's equation in the unit square, the model problem of Chapter 2, and so will not be discussed further.

We consider only the results for $\alpha = \frac{\pi}{3}$ and $\alpha = \frac{\pi}{12}$.

Tables 3.8 and 3.12 indicate that point SOR theory provides a reasonable estimate of the experimental results only for large values of $\alpha$ and small values of $h^{-1}$. Unlike the point SOR method in the equilateral-triangular region, it cannot be said that the theory holds for practical purposes. For this reason we cannot meaningfully compare its theoretical asymptotic rate of convergence with the rates of convergence of the block SOR methods.

A comparison of the estimated number of iterations required for convergence (using either of the two theories) shows that S2LOR is a factor of about 1.4 faster than SLOR, whilst SPOR and SLOR have the same rate of convergence. This is equivalent to the results obtained for the square region. A comparison of the actual number of iterations required indicates that S2LOR converges about 1.4 times as fast as SLOR in practice (as well as in theory) whilst SPOR is a factor of 1.2 faster than SLOR. The discrepancy between theory and practice is due to the fact that the behaviour of SPOR is described much better by theory 1 than theory 2, but the reverse is true for SLOR. The number of iterations required for S2LOR lies between the two
predicted values, closer to (equal to, for small values of $h^{-1}$) theory 2.

We consider the amount of work needed to perform each iteration.

For $\alpha \neq \frac{\pi}{2}$ the work consists of

- 10 additions
- 9 multiplications
- 1 convergence test for point, line and two-line SOR,

and

- 12 additions
- 11 multiplications
- 1 convergence test for peripheral SOR.

(3.19)

When these figures are combined with the results for number of iterations required, the following times per point for convergence are obtained.

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>408</td>
<td>340</td>
<td>238</td>
<td>328</td>
</tr>
<tr>
<td>10</td>
<td>918</td>
<td>680</td>
<td>476</td>
<td>656</td>
</tr>
<tr>
<td>20</td>
<td>1836</td>
<td>1360</td>
<td>884</td>
<td>1271</td>
</tr>
<tr>
<td>40</td>
<td>3556</td>
<td>2586</td>
<td>1700</td>
<td>2074</td>
</tr>
</tbody>
</table>

Table 3.17 gives the same information as Table 3.16, but relative to point SOR.
TABLE 3.17
\[ \alpha = \frac{\pi}{3} \]. Relative work per point

<table>
<thead>
<tr>
<th>( h^{-1} )</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1.000</td>
<td>0.833</td>
<td>0.583</td>
<td>0.804</td>
</tr>
<tr>
<td>10</td>
<td>1.000</td>
<td>0.741</td>
<td>0.519</td>
<td>0.715</td>
</tr>
<tr>
<td>20</td>
<td>1.000</td>
<td>0.741</td>
<td>0.481</td>
<td>0.692</td>
</tr>
<tr>
<td>40</td>
<td>1.000</td>
<td>0.760</td>
<td>0.481</td>
<td>0.587</td>
</tr>
<tr>
<td>Average</td>
<td>1.000</td>
<td>0.769</td>
<td>0.516</td>
<td>0.700</td>
</tr>
</tbody>
</table>

The corresponding data for \( \alpha = \frac{\pi}{12} \) is given by tables 3.18 and 3.19.

TABLE 3.18
\[ \alpha = \frac{\pi}{12} \]. Work per point

<table>
<thead>
<tr>
<th>( h^{-1} )</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>544</td>
<td>340</td>
<td>272</td>
<td>328</td>
</tr>
<tr>
<td>10</td>
<td>1428</td>
<td>714</td>
<td>510</td>
<td>738</td>
</tr>
<tr>
<td>20</td>
<td>3706</td>
<td>1462</td>
<td>986</td>
<td>1517</td>
</tr>
<tr>
<td>40</td>
<td>8908</td>
<td>2890</td>
<td>2006</td>
<td>3034</td>
</tr>
</tbody>
</table>
The preceding tables indicate that the block SOR methods are more efficient than point SOR, and the improvement in efficiency increases rapidly as $h^{-1}$ increases. While the former conclusion is undoubtedly true, the latter is due to the very poor convergence of the point method for large values of $h^{-1}$ rather than very good convergence of the block methods. In the first pair of tables ($a = \frac{n}{3}$) both S2LOR and SPOR improve their efficiency a little relative to SLOR, but in the second pair of tables ($a = \frac{n}{12}$) there are no such trends. The average ratio figures of tables 3.17 and 3.19, which would be meaningless if there were sharp trends in the relative efficiencies, in fact give a good indication of the relative merits of the block SOR methods. S2LOR is clearly the most efficient of the three methods. SLOR and SPOR are roughly similar; for $a = \frac{n}{12}$ and $a = \frac{n}{2}$ (see table 2.13), SLOR is superior whilst for $a = \frac{n}{3}$, SPOR is the more efficient method.
CHAPTER 4

TECHNIQUES FOR THE SOLUTION OF

MILDLY NON-LINEAR EQUATIONS
4.1 INTRODUCTION

This chapter deals with the solution of mildly non-linear partial
differential equations in square regions. Two typical equations are studied.
The first is
\[ \nabla^2 \phi = 0.1 \, e^\phi ; \]  \hfill (4.1)
equations of this form occur in vortex problems, the theory of the space
charge of electricity around a glowing wire, and the nebular theory for
the distribution of mass of gaseous interstellar material. The second
equation to be studied is
\[ \nabla^2 \phi = \phi^2 , \]  \hfill (4.2)
which occurs frequently in the study of diffusion controlled chemical
reactions.

The efficiency of various strategies in carrying out the solution
process is examined. Point and block SOR methods are applied to the problem,
and various adaptive strategies for determining the optimum SOR acceleration
parameter are compared. The preconditioning technique is also used to solve
the equations.
4.2 INNER/OUTER ITERATION STRATEGIES

A well-known method of solving mildly non-linear equations is to use an iterative procedure, each step of which involves the iterative solution of a system of linear algebraic equations. Figure 4.1 describes this process in flowchart form. Let the finite-difference representation of the non-linear equation be

$$A_\mathbf{z} = f(\mathbf{z})$$  \hspace{1cm} (4.3)

where \( f \) is some function of \( \mathbf{z} \), and let the inner iteration be performed by the process of the form

$$\mathbf{z}_{(n+1)}^{(i,j)} = M \mathbf{z}_{(n)}^{(i)} + d^{(i)}$$  \hspace{1cm} (4.4)

In the flowchart, \( \mathbf{z}_{(i,j)}^{(i,j)} \) denotes the iterate at the \( i^{th} \) outer iteration, \( j^{th} \) inner iteration. \( \mathbf{z}_{(i,*)}^{(i)} \) is the value of the iterate at the completion of the \( i^{th} \) outer iteration, \( d^{(i)} \) is the constant vector of the inner iteration process for the \( i^{th} \) outer iteration. The function \( g \) is defined by

$$g(\mathbf{z}) = (I - MA^{-1})f(\mathbf{z})$$  \hspace{1cm} (4.5)

Note that the flowchart represents the logical structure of the procedure and not its most efficient implementation.

The aim in choosing the inner/outer loop strategy is to minimise the overall computational effort. Each inner loop requires one sweep of the iterative process (4.4), whilst each outer loop requires the evaluation of the right-hand-side of the p.d.e. for each mesh point.

(In practice, the vectors \( d^{(i)} \) are not calculated explicitly. The quantity \( f(\mathbf{z}_{(i-1,*)}^{(i-1,*)}) \) is evaluated at the start of the \( i^{th} \) iteration (box 2 of the flowchart), and the vector \( d^{(i)} \) computed as part of the inner iterative process, which is not implemented in the simple explicit form indicated in box 4 of the flowchart).

It is not satisfactory to minimise the number of inner iterations at the expense of the outer, or vice versa, it is necessary to strike a balance between them.
1. Initialise \( z^{(0,) \ast} \)
   \( i \leftarrow 0 \)

2. \( i + i + 1 \)
   \( d^{(i)}(i) \leftarrow g(z^{(i-1, \ast)}) \)

3. \( j = 0 \)
   \( z^{(i, 0)} + z^{(i-1, \ast)} \)

4. \( j + j + 1 \)
   \( z^{(i,j)} + MZ^{(i,j-1)} + d^{(i)} \)

5. \( z^{(i,j)} = z^{(i,j-1)} \)?
   \( (1) \)
   YES
   NO

6. \( z^{(i, \ast)} = z^{(i-1, \ast)} \)?
   \( (1) \)
   YES
   NO

(1) To within specified tolerance

**FIGURE 4.1**
The first few iterations represent fairly crude approximations to the mildly non-linear equation and it might be thought not worthwhile to solve them very accurately. This approach can be implemented by initially choosing a crude convergence criterion for the inner iteration (box 5 of the flowchart) and progressively increasing its stringency (up to a suitable limit) during the solution process. Numerical experiments indicate that this strategy is actually very inefficient, since it produces a large number of outer iterations, each of which contains a very small number (often 1) of inner iterations. In contrast, the use of a fixed, stringent test throughout leads to a much smaller number of outer iterations, and very few more inner iterations.

An alternative approach is to use a fixed number of inner iterations \( k \), say, in each outer iteration, thereby eliminating the work of testing for convergence in each inner iteration. Experiments show that this strategy is only effective with a good choice of \( k \). If \( k \) is too small, the number of outer iterations is overly large, whilst if \( k \) is too big, a substantial number of inner iterations may be performed after the problem has converged because there are no tests on the inner loop to halt the process.

On the basis of these experiments, two reasonably effective strategies were selected for further study, inner convergence tests of constant strictness, and a fixed number (5) of inner iterations per outer loop. These are referred to as strategy 1 and strategy 2 respectively.
4.3 EXPERIMENTAL RESULTS FOR POINT AND BLOCK SOR METHODS

We solve two different problems in the unit square, using for the inner iterations the various SOR methods described in Chapter 2. Problem I is the solution of equation (4.1) with zero boundary conditions. Problem II is the solution of equation (4.2) with boundary conditions unity along one side of the square, and zero elsewhere. The initial approximation to the solution is zero in all cases.

The optimum values of the SOR parameter \( \omega \) are determined experimentally correct to two significant figures. The optimum value is defined as that which minimises the number of inner iterations required. Outer iterations are ignored in this definition, since in strategy 1 methods their number is generally the same for any given problem whatever the value of \( \omega \), and in strategy 2 methods their number is simply a multiple of the number of inner iterations. Convergence is deemed to have occurred when the maximum absolute difference between corresponding elements of successive iterates is less than \( 5 \times 10^{-6} \).

Experimental results are presented in the following tables.
TABLE 4.1
Mildly non-linear equations, problem I. SOR methods with optimum $\omega$

<table>
<thead>
<tr>
<th>Method</th>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>Outer iterations</th>
<th>Inner iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point SOR</td>
<td>20</td>
<td>1</td>
<td>3</td>
<td>33</td>
<td>1.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>7</td>
<td>35</td>
<td>1.80</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>3</td>
<td>44</td>
<td>1.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>10</td>
<td>50</td>
<td>1.82</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>3</td>
<td>57</td>
<td>1.86</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>15</td>
<td>75</td>
<td>1.85-1.86</td>
</tr>
<tr>
<td>SLOR</td>
<td>20</td>
<td>1</td>
<td>3</td>
<td>26</td>
<td>1.66-1.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td>25</td>
<td>1.67</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>3</td>
<td>34</td>
<td>1.76-1.78</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>7</td>
<td>35</td>
<td>1.77</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>3</td>
<td>42</td>
<td>1.81-1.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>9</td>
<td>45</td>
<td>1.82</td>
</tr>
<tr>
<td>S2LOR</td>
<td>20</td>
<td>1</td>
<td>3</td>
<td>21</td>
<td>1.55-1.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>4</td>
<td>20</td>
<td>1.60-1.61</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>2</td>
<td>22</td>
<td>1.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td>25</td>
<td>1.68</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>2</td>
<td>27</td>
<td>1.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>8</td>
<td>40</td>
<td>1.72</td>
</tr>
<tr>
<td>SPOR</td>
<td>20</td>
<td>1</td>
<td>3</td>
<td>23</td>
<td>1.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td>25</td>
<td>1.66-1.68</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>3</td>
<td>29</td>
<td>1.77</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>7</td>
<td>35</td>
<td>1.75-1.80</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>3</td>
<td>36</td>
<td>1.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>8</td>
<td>40</td>
<td>1.82</td>
</tr>
</tbody>
</table>
### TABLE 4.2

Mildly non-linear equations, problem II. SOR methods with optimum $\omega$

<table>
<thead>
<tr>
<th>Method</th>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>Outer iterations</th>
<th>Inner iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point SOR</td>
<td>20</td>
<td>1</td>
<td>4</td>
<td>86</td>
<td>1.73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>10</td>
<td>50</td>
<td>1.73-1.76</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>4</td>
<td>122</td>
<td>1.82-1.83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>13</td>
<td>65</td>
<td>1.81</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>4</td>
<td>160</td>
<td>1.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>18</td>
<td>90</td>
<td>1.86</td>
</tr>
<tr>
<td>SLOR</td>
<td>20</td>
<td>1</td>
<td>4</td>
<td>58</td>
<td>1.65-1.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>7</td>
<td>35</td>
<td>1.63-1.69</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>4</td>
<td>81</td>
<td>1.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>9</td>
<td>45</td>
<td>1.74-1.76</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>4</td>
<td>104</td>
<td>1.81</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>11</td>
<td>55</td>
<td>1.81</td>
</tr>
<tr>
<td>S2LOR</td>
<td>20</td>
<td>1</td>
<td>4</td>
<td>45</td>
<td>1.54-1.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td>25</td>
<td>1.55</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>4</td>
<td>63</td>
<td>1.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>7</td>
<td>35</td>
<td>1.65-1.66</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>4</td>
<td>82</td>
<td>1.73-1.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>9</td>
<td>45</td>
<td>1.73-1.74</td>
</tr>
<tr>
<td>SPOR</td>
<td>20</td>
<td>1</td>
<td>4</td>
<td>59</td>
<td>1.65</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>7</td>
<td>35</td>
<td>1.63-1.67</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>4</td>
<td>84</td>
<td>1.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>9</td>
<td>45</td>
<td>1.75</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>4</td>
<td>110</td>
<td>1.80</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>12</td>
<td>60</td>
<td>1.82</td>
</tr>
</tbody>
</table>
The experimental optimum values of \( w \) in the preceding tables agree fairly well with the theoretical values established in Chapter 2 for Laplace's equation in the same region. The agreement is very good for problem II but variable for problem I, particularly for strategy 2. By its very nature, strategy 2 is insensitive to small changes in \( w \), so that a near-minimum number of iterations is achieved over a range of values of \( w \). Within this range, a small change in the behaviour of the iterative process can produce a minimum number of iterations which may or may not correspond to the theoretical optimum \( w \).

For problem I, strategy 2 generally needs a few more inner iterations than strategy 1, and more outer iterations. However, an inner iteration for strategy 2 requires less work than one for strategy 1 because it does not include a convergence test. Hence the relative efficiency of the two strategies cannot be deduced from the iteration count alone. For problem II, strategy 2 needs considerably less inner iterations than strategy 1, but because the number of inner iterations is large, it needs many more outer iterations. Here again the number of iterations required does not give a clear picture of the relative efficiency of the strategies.

It is noteworthy that the relative behaviour of the strategies is very different in the two problems considered. It is suggested that this is because problem I involves only zero boundary conditions, whereas problem II has a non-zero boundary condition.

In the notation of Figure 4.1, the initial estimate \( z^{(0,*)} \) of the solution is set to zero and the first estimate \( d^{(1)} \) of the right-hand side is calculated from this and from the boundary conditions. If the boundary conditions are non-zero, this value of \( d^{(1)} \) is a poor estimate of the right-hand side of the non-linear equations since it is calculated from a vector whose value is not influenced by the boundary conditions at all. Hence the set of linear equations in the first outer iteration gives a poor
approximation to the set of non-linear equations we wish to solve. Strategy 1 methods spend a large number of inner iterations in solving this first approximation, effort which is largely wasted. Strategy 2 methods perform only five iterations before improving the value of the right-hand side and hence making the set of linear equations a better estimate of the set of non-linear equations. However, if the boundary conditions of the problem are zero, the vector $d^{(1)}$ is a much better estimate of the right-hand side of the non-linear equations, so that there is less difference in the behaviour of the two strategies.

If this theory is true, we would expect that the discrepancy between the two strategies (at the optimum values of $\omega$) would be reduced in problem II if the boundary conditions were reduced. That this does happen is illustrated by Table 4.3 below. (For simplicity, only the numbers of inner iterations are given). The boundary conditions are $\phi=0$ along three sides of the square, and $\phi=u$ along the fourth side, where $u$ takes the values 1, 0.5 and 0.1.
**TABLE 4.3**

Inner iterations required to solve equation (4.2) with various boundary conditions

| Method | $h^{-1}$ | $u=1.0$ | | $u=0.5$ | | $u=0.1$ |
|--------|----------|---------|--------|---------|--------|
|        | Strategy 1 | Strategy 2 | Strategy 1 | Strategy 2 | Strategy 1 | Strategy 2 |
| Point SOR | 20 | 86 | 50 | 69 | 45 | 45 |
|          | 30 | 122 | 65 | 98 | 65 | 60 |
|          | 40 | 160 | 90 | 126 | 85 | 74 |
| SLOR    | 20 | 58 | 35 | 47 | 30 | 29 |
|          | 30 | 81 | 45 | 65 | 45 | 41 |
|          | 40 | 104 | 55 | 74 | 55 | 52 |
| S2LOR   | 20 | 45 | 25 | 38 | 25 | 25 |
|          | 30 | 63 | 35 | 42 | 35 | 34 |
|          | 40 | 82 | 45 | 62 | 45 | 39 |
| SPOR    | 20 | 59 | 35 | 48 | 35 | 31 |
|          | 30 | 84 | 45 | 65 | 45 | 43 |
|          | 40 | 110 | 60 | 80 | 60 | 54 |
We now consider the relative efficiency of the various combinations of SOR methods and inner/outer iteration strategies. Table 4.4 gives the execution times per point for the inner iterations. Execution times are calculated using the weighting scheme developed in Chapter 2.

**TABLE 4.4**

Execution times per point (inner iterations)

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy 1</td>
<td>20.0</td>
<td>20.0</td>
<td>23.5</td>
<td>27.0</td>
</tr>
<tr>
<td>Strategy 2</td>
<td>18.5</td>
<td>18.5</td>
<td>22.0</td>
<td>25.5</td>
</tr>
</tbody>
</table>

Outer iterations involve a function evaluation and a convergence test for each point. The function evaluations take the form

\[-h^2 \times 0.1 \exp(z_{i,j})\]  \hspace{1cm} (4.6)

for Problem I and

\[-h^2 \times z_{i,j}^2\]  \hspace{1cm} (4.7)

for Problem II. The constant \(-h^2\) can be calculated once and stored, so that expression (4.7) involves 2 multiplications and hence has an execution time of 5. The execution time for equation (4.6) is estimated as 20. We note that if the most efficient forms of the block SOR methods are used, the inner iterations are carried out using a transformation of the vector \(\mathbf{z}\). Thus it will be necessary to perform the inverse transformation to obtain \(\mathbf{z}\) before doing the function evaluation at the start of each outer iteration. This requires an extra multiplication per point. The convergence test differs from that used in the inner iteration, since the vector of differences between successive iterates is not calculated explicitly as part of the iterative process. Hence the outer-loop convergence test has the form
\[ |z_{i,j}^{(n+1)} - z_{i,j}^{(n)}| < \varepsilon. \] (4.8)

This involves 2 or 3 subtractions so we assign it an (average) execution time of 2.5. The overall execution times for the outer iterations are given in Table 4.5.

**TABLE 4.5**

<table>
<thead>
<tr>
<th>Problem</th>
<th>Point SOR</th>
<th>Block SOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem I</td>
<td>22.5</td>
<td>25</td>
</tr>
<tr>
<td>Problem II</td>
<td>7.5</td>
<td>10</td>
</tr>
</tbody>
</table>

Tables 4.6 and 4.7 show clearly the differing effects of the two strategies. In Table 4.6 (Problem I), strategy 2 methods generally require 10-25% more computer time than the corresponding strategy 1 methods. The difference between the two strategies increases as \( h^{-1} \) increases. In Table 4.7 (Problem II), strategy 2 methods need 40-50% less computer time than the corresponding strategy 1 methods. Here again, the difference between the two strategies increases as \( h^{-1} \) increases.

The various methods and strategies are ranked in order of efficiency in Table 4.8.
**TABLE 4.6**

Mildly non-linear equations, Problem I. Work per point

<table>
<thead>
<tr>
<th>Method</th>
<th>Point</th>
<th>SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>h⁻¹</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>727.5</td>
<td>805</td>
<td>595</td>
<td>587.5</td>
<td>568.5</td>
</tr>
<tr>
<td>30</td>
<td>947.5</td>
<td>1150</td>
<td>755</td>
<td>822.5</td>
<td>567</td>
</tr>
<tr>
<td>40</td>
<td>1207.5</td>
<td>1725</td>
<td>915</td>
<td>1055.5</td>
<td>684.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Point</th>
<th>SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>h⁻¹</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>727.5</td>
<td>805</td>
<td>595</td>
<td>587.5</td>
<td>568.5</td>
</tr>
<tr>
<td>30</td>
<td>947.5</td>
<td>1150</td>
<td>755</td>
<td>822.5</td>
<td>567</td>
</tr>
<tr>
<td>40</td>
<td>1207.5</td>
<td>1725</td>
<td>915</td>
<td>1055.5</td>
<td>684.5</td>
</tr>
</tbody>
</table>
### TABLE 4.7

Mildly non-linear equations, Problem II. Work per point

<table>
<thead>
<tr>
<th>Method</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1750</td>
<td>1000</td>
<td>1200</td>
<td>717.5</td>
</tr>
<tr>
<td>30</td>
<td>2470</td>
<td>1300</td>
<td>1660</td>
<td>922.5</td>
</tr>
<tr>
<td>40</td>
<td>3230</td>
<td>1800</td>
<td>2120</td>
<td>1127.5</td>
</tr>
</tbody>
</table>
### TABLE 4.8

Mildly non-linear equations. Comparative efficiency of various methods and strategies

<table>
<thead>
<tr>
<th>Efficiency ranking</th>
<th>Problem I</th>
<th>Problem 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Method</td>
<td>Strategy</td>
</tr>
<tr>
<td>1</td>
<td>S2LOR</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>S2LOR</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>SLOR</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>SLOR</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>SPOR</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Point SOR</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>SPOR</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>Point SOR</td>
<td>2</td>
</tr>
</tbody>
</table>

In Problem I, the influence of the strategy is secondary to the influence of the method. The ranking of the methods is as for the equivalent linear problem (the model problem of Chapter 2), with strategy 1 being slightly better than strategy 2. The exception to this rule is that point SOR, strategy 1 and SPOR, strategy 2 are interchanged; however, the difference between them is very small. The most efficient combination of method and strategy requires about 50% less work than the worst combination, so that it is worth exercising considerable care in the choice of approach to such a problem.

Results for Problem II show that the influence of the strategy is
paramount in this case, so that all the strategy 2 methods are better than all the strategy 1 methods. Within each strategy, the ranking of the methods is the same as for the equivalent linear problem. Here the most efficient combination of method and strategy needs only a third of the computational effort of the worst combination.

Whilst it is impossible to make firm statements about the best technique for solving any mildly non-linear equation on the basis of a limited number of experiments, the following observations can be made:

1. Two-line SOR is consistently the best of the four SOR methods studied in relation to our two trial mildly non-linear problems. There is no apparent reason why this method should not be the most efficient for any such problem.

2. If the right-hand side of the p.d.e. requires a large amount of computational effort to evaluate, it is desirable to minimise the number of times it has to be computed, i.e. to minimise the number of outer iterations. The number required by strategy 1 methods depends on the nature of the problem but not, apparently, its size. On the other hand, the number needed by strategy 2 methods increases sharply as $h^{-1}$ increases. Hence the more complicated the right-hand side and the larger the number of mesh points, the more strategy 1 is to be favoured.

3. The use of strategy 2 minimises the number of inner iterations when the boundary conditions are non-zero. It is not clear whether the improvement of strategy 2 over strategy 1 depends on the absolute value of the boundary conditions, or the difference between values on different parts of the boundary. The results of Table 4,3 could indicate either reason. Probably both factors have some influence.
Certain mildly non-linear equations can be linearised by incorporating the right-hand side into the finite-difference equation. Equation (4.1) is unsuitable for this treatment, but equation (4.2) can be linearised. The basic form of the finite-difference equation at the mesh point \((i,j)\) is

\[
4z_{i,j} - z_{i,j+1} - z_{i,j-1} - z_{i+1,j} - z_{i-1,j} = -h^2 z_{i,j}.
\]

This can be linearised to give the equation

\[
(4+h^2 z_{i,j})z_{i,j} - z_{i,j+1} - z_{i,j-1} - z_{i+1,j} - z_{i-1,j} = 0.
\]

The inner iteration process, which consists of solving sets of these equations, can be carried out by any of the standard iterative techniques such as point and block Jacobi, Gauss-Seidel and SOR methods. As an example we consider the point Jacobi method. We use the notation of Figure 4.1 with \(I\) denoting the outer iteration subscript and \(J\) the inner iteration subscript. The point Jacobi method has the form

\[
(4+h^2 z_{i,j})z_{i,j} - z_{i,j+1} - z_{i,j-1} - z_{i+1,j} - z_{i-1,j} = 0.
\]

Thus instead of each outer iteration solving the same set of finite-difference equations with a different right-hand side in each case, (as in the previous section), each outer iteration solves a different set of finite-difference equations with the same right-hand side, namely zero.

We present experimental results for the solution of sets of equations of the form (4.9') in the unit square, the boundary conditions being zero along three sides of the square, and unity along the fourth. Point and block SOR methods are used in combination with inner/outer iteration strategies 1 and 2. The values of the SOR parameter \(\omega\) which minimise the number of inner iterations in each case are determined correct to two decimal places. The iterative processes are deemed to have converged when
the maximum absolute difference between corresponding elements of successive iterates is less than $5 \times 10^{-6}$.

The results for the linearised equations are very similar to those for the equations without linearisation (Table 4.2). In most cases the linearised equations require a few less inner iterations for strategy 1 methods, but need the same number or slightly more for strategy 2 methods. We conclude that linearising the finite-difference equations has not significantly affected the rate of convergence of the SOR methods.

The optimum values of $\omega$ match those for the equations without linearisation exactly in the majority of cases. This is despite the fact that the linearised equations are different in every outer iteration and hence the optimum $\omega$ will change. However, the changes in the equations are likely to be small, especially if $h$ is small.

The question arises as to whether linearising the finite-difference equations saves any computational effort. The effect of linearising is to replace the calculation of the right-hand side at the start of each outer iteration by the calculation of the coefficients of the finite-difference equation. In point SOR it is only necessary to evaluate

$$\frac{1}{4 + h^2 z_{i,j}}$$

for all mesh points $(i,j)$. Block methods, however, require the evaluation of several expressions for each mesh point to produce the parameters associated with the solution of the blocks of equations. The normalised triangular decomposition algorithms usually used for this purpose are unsuitable here because the calculation of the parameters is very complicated. In particular, it involves evaluating a square root for each mesh point, which in the problem being considered is more work than the function evaluation it replaces.
### Table 4.9

Mildly non-linear equations, problem II SOR methods
(linearised) with optimum $\omega$

<table>
<thead>
<tr>
<th>Method</th>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>Outer Iterations</th>
<th>Inner Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Point SOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>4</td>
<td>84</td>
<td></td>
<td>1.73</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>50</td>
<td></td>
<td>1.74-1.76</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>4</td>
<td>118</td>
<td></td>
<td>1.81-1.83</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>14</td>
<td>70</td>
<td></td>
<td>1.82-1.83</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>4</td>
<td>151</td>
<td></td>
<td>1.86</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>18</td>
<td>90</td>
<td></td>
<td>1.86</td>
</tr>
<tr>
<td><strong>SLOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>4</td>
<td>56</td>
<td></td>
<td>1.65-1.66</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6</td>
<td>30</td>
<td></td>
<td>1.66</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>4</td>
<td>79</td>
<td></td>
<td>1.75</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>50</td>
<td></td>
<td>1.75-1.77</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>4</td>
<td>100</td>
<td></td>
<td>1.81</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11</td>
<td>55</td>
<td></td>
<td>1.81</td>
</tr>
<tr>
<td><strong>S2LOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>4</td>
<td>43</td>
<td></td>
<td>1.55</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>6</td>
<td>30</td>
<td></td>
<td>1.51-1.62</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>4</td>
<td>61</td>
<td></td>
<td>1.67</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td>35</td>
<td></td>
<td>1.67</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>4</td>
<td>79</td>
<td></td>
<td>1.73</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>50</td>
<td></td>
<td>1.75-1.76</td>
</tr>
<tr>
<td><strong>SPOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>4</td>
<td>58</td>
<td></td>
<td>1.65</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td>35</td>
<td></td>
<td>1.65-1.68</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>4</td>
<td>82</td>
<td></td>
<td>1.75</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>50</td>
<td></td>
<td>1.75-1.76</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>4</td>
<td>106</td>
<td></td>
<td>1.80-1.81</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>12</td>
<td>60</td>
<td></td>
<td>1.81</td>
</tr>
</tbody>
</table>
We use instead the Gaussian elimination algorithms presented in Chapter 2. These require one more multiplication per point in the inner iteration than the corresponding normalised algorithms, but considerably less arithmetic operations in the outer iteration.

The linearisation process does save one addition per point in the inner iteration because the right-hand side of the finite-difference equations is zero and therefore need not be added in.

The operations required by each of the four SOR methods under consideration is as follows:

Point SOR,

outer iteration,
1 addition
1 multiplication
1 division
1 convergence test

inner iteration,
6 additions
5 multiplications
1 convergence test (strategy 1 only)

Line SOR,

outer iteration,
2 additions
3 multiplications
2 divisions
1 convergence test

inner iteration,
5 additions
6 multiplications
1 convergence test (strategy 1 only)
Two-line SOR,

outer iteration,

  5 additions,
  7 multiplications
  2 divisions
  1 convergence test

inner iteration,

  6 additions
  7 multiplications
  1 convergence test (strategy 1 only)

and Peripheral SOR,

outer iteration,

  3 additions
  7 multiplications
  2 divisions
  1 convergence test

inner iteration,

  7 additions
  8 multiplications
  1 convergence test (strategy 1 only)

Note that a convergence test in an outer iteration has an execution
time of 2.5 but a convergence test in an inner iteration has an
execution time of 1.5. The relative execution times for the four
methods are shown in Table 4.10.

To compare the overall efficiency of the methods we combine this
data with the results of Table 4.9.
TABLE 4.10
Mildly non-linear equations, Problem II (linearised). Execution times per point (outer and inner iterations)

<table>
<thead>
<tr>
<th>Method</th>
<th>Outer iteration</th>
<th>Inner iteration, strategy 1</th>
<th>Inner iteration, strategy 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Point SOR</td>
<td>SLOR</td>
<td>S2LOR</td>
</tr>
<tr>
<td>Outer iteration</td>
<td>9.5</td>
<td>19</td>
<td>32</td>
</tr>
<tr>
<td>Inner iteration, strategy 1</td>
<td>20</td>
<td>21.5</td>
<td>25</td>
</tr>
<tr>
<td>Inner iteration, strategy 2</td>
<td>18.5</td>
<td>20</td>
<td>23.5</td>
</tr>
</tbody>
</table>

TABLE 4.11
Mildly non-linear equations, Problem II (linearised). Work per point

<table>
<thead>
<tr>
<th>Method</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>h^{-1} 1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>1718</td>
<td>1020</td>
<td>1280</td>
<td>714</td>
</tr>
<tr>
<td>30</td>
<td>2398</td>
<td>1428</td>
<td>1774.5</td>
<td>1190</td>
</tr>
<tr>
<td>40</td>
<td>3058</td>
<td>1836</td>
<td>2226</td>
<td>1309</td>
</tr>
</tbody>
</table>

A comparison of this table with Table 4.7, which gives execution times for the same problem and methods without linearisation, shows that the linearisation technique generally produces a trivial saving of the effort associated with the point SOR method, but increases the effort required for the block methods. On the basis of this experiment, linearisation cannot be said to be worthwhile. It is unlikely that the
technique could ever produce a saving of effort when used in conjunction with one of the block SOR methods, since the work done in re-calculating the coefficients of the finite-difference equations at the start of each outer iteration is substantial. A non-trivial saving of effort might be achieved in some problems when the technique is used in conjunction with the point SOR method, but this method is relatively inefficient anyway.
4.5 ADAPTIVE TECHNIQUES FOR THE DETERMINATION OF THE OPTIMUM OVER-RELAXATION FACTOR OF THE SOR METHOD

The effectiveness of the SOR methods in solving elliptic p.d.e.'s depends very much on the value of the over-relaxation factor used being close to the optimum. It is possible to determine this parameter theoretically for some SOR methods in rectangular regions, but this is a very limited class of problems. We seek some method of determining the optimum parameter $\omega_b$ which is more generally applicable.

It is possible to compute the spectral radius of the Jacobi iteration matrix by means of the power method, and hence calculate $\omega_b$ by the usual formula. This gives a very accurate value for $\omega_b$, but the power method converges very slowly. Hence this technique is worthwhile if many problems are to be solved in the same region but requires a disproportionately large amount of computational effort if the problem is only to be solved once.

Young (1955) describes methods of estimating $\omega_b$ by calculating the optimum parameter for a rectangle which approximates the region of solution. He considers the use of a circumscribing rectangle and a rectangle whose area is the same as that of the region of solution. He shows that these approaches give accurate results in some cases but unacceptably bad ones in others. A more consistent method would be preferable.

A number of adaptive techniques have been developed whereby an initial estimate of $\omega_b$ is successively modified as the iterative solution of the p.d.e. proceeds. The norms of the displacement vectors are used to estimate the spectral radius of the Jacobi matrix and hence improve the estimate of the over-relaxation parameter. We consider three such methods, those proposed by Kulsrud (1961), pp.184-187, Carré (1961), pp.73-78 and
Reid (1966), pp.200-204. These techniques have been shown to be successful in the solution of linear p.d.e.'s, where the iterate converges smoothly. We shall be concerned with their application to mildly non-linear p.d.e.'s, where discontinuities are introduced into the (inner) iterate by the change in the right-hand side of the finite-difference equations at the start of each outer iteration.

The theory of the three methods is presented here in outline only. The reader is referred to the original publications for full descriptions of the methods.

4.5.1 Kulsrud's Technique

The theory of Kulsrud's technique is as follows. The SOR iterative method has the form

$$z^{(n+1)} = M_w z^{(n)} + d, \quad n=0,1,2,...$$  \hspace{1cm} (4.12)

where $M_w$ is the iteration matrix of the SOR method and $z^{(0)}$ is arbitrary. It follows that $z^{(n+1)}$ can be written as

$$z^{(n+1)} = M_w^{n+1} z^{(0)} + \sum_{i=0}^{n} M_w^i d, \quad n \geq 1.$$  \hspace{1cm} (4.13)

The difference between successive iterates (i.e. the displacement vector), is given by

$$z^{(n+1)} - z^{(n)} = M_w^{n} (M_w z^{(0)} + d - z^{(0)}), \quad n \geq 1$$

$$= M_w^n x, \quad \text{say.}$$  \hspace{1cm} (4.14)

If $M_w$ has linearly independent eigenvectors $v_1, v_2, ..., v_N$, with corresponding real eigenvalues $\lambda_1, \lambda_2, ..., \lambda_N$, then $x$ can be expressed as

$$x = \sum_{i=1}^{N} a_i v_i$$  \hspace{1cm} (4.15)

for some $a_i, i=1,2, ..., N$. Hence by definition we have
If \( \lambda_1 \) is the eigenvalue of largest modulus, then

\[
||z^{(n+1)} - z^{(n)}|| = ||a_1 \lambda_1^n v_1||
\]

\[
= ||a_1| |\lambda_1|^n | |v_1||
\]  

(4.17)

for some norm \( ||.|| \).

Then the ratio of the norms of two successive displacement vectors is

\[
\frac{||z^{(n+1)} - z^{(n)}||}{||z^{(n)} - z^{(n-1)}||} = |\lambda_1| = \overline{\lambda}, \text{ say.}
\]  

(4.18)

From this approximation, an estimate can be made of \( \overline{\lambda} \), the spectral radius of the Jacobi matrix, by

\[
\overline{\lambda} = \frac{\lambda + \omega - 1}{\omega \lambda} .
\]  

(4.19)

An improved estimate of \( \omega_b \) can then be made by the usual formula,

\[
\omega_b = \frac{2}{1 + \sqrt{1 - \overline{\lambda}^2}} .
\]  

(4.20)

Kulsrud's technique is to make an initial estimate of \( \omega_b \), which is then improved at intervals using the above procedure until the estimate has converged to the desired accuracy. This estimate is used throughout the remainder of the iterative solution of the equations.

Let the \( k \text{th} \) estimate of \( \omega_b \) be \( \omega^{(k)} \) and the \( k \text{th} \) estimate of \( \overline{\lambda}(M_{\omega_b}) \) be \( \overline{\lambda}^{(k)} \). (Note that \( \overline{\lambda}^{(k)} \) is the value calculated after a number of iterations with \( \omega = \omega^{(k)} \).) It can be shown that while

\[
\overline{\lambda}^{(k)} \neq \omega^{(k)} - 1 ,
\]  

(4.21)

then \( \omega^{(k)} \), \( k=1,2,\ldots \) is a monotonically increasing sequence.

Eventually, we will have

\[
\omega^{(p)} \geq \omega_b > \omega^{(p-1)}
\]  

(4.22)
for some \( k = p \), and from SOR theory we know that

\[
\lambda(p) = \omega(p) - 1 .
\] (4.23)

It can be shown that \( \omega(p) \) is the limit of the \( \omega(k) \) sequence, i.e. as soon as the estimate \( \omega(k) \) reaches or exceeds \( \omega_b \), the sequence converges.

It is important that the first estimate of \( \omega_b \) should be less than the optimum. Once the estimate \( \omega(k) \) is greater than \( \omega_b \), the method described for estimating \( \lambda \) is not very good, since all the eigenvalues \( \lambda_i \) are complex, and the results are unpredictable.

Kulsrud gives no indication of how many iterations should be performed between making new estimates of \( \omega_b \). If the number of iterations is too small, the estimates of \( \lambda \) (and therefore of \( \omega_b \)) will be poor. On the other hand, if the number is too large, a large proportion of the SOR iteration will be carried out with values of \( \omega \) less than the optimum.

Experimental results are presented here for Kulsrud's method applied to the solution of problems I and II (as defined earlier), by the various combinations of SOR methods and inner/outer iteration strategies described in Section 4.3.

The initial estimate of \( \omega_b \) in every case is taken as 1.35, well below the values obtained experimentally. New estimates are calculated every 5 inner iterations, so that in strategy 2 methods, a new estimate of \( \omega_b \) is made at the end of each outer iteration. The norm used in equation (4.18) is the \( \| \cdot \|_1 \) norm, chosen because it requires very little work to compute Kulsrud does not suggest a convergence test to be applied to the \( \omega(k) \) sequence. We use

\[
\frac{\left| \omega(k+1) - \omega(k) \right|}{2 - \omega(k+1)} < 0.05
\] (4.24)

as our criterion for stopping. This test is used by Reid (1966) when comparing his own method with those of Kulsrud and Carré.
In the following tables, we record the number of inner iterations required for the \( w(k) \) sequence to converge (\( n_{\omega_b} \)), the converged value \( (\omega_b(p)) \), and the number of outer and inner iterations required for the equations to converge. (The equations are deemed to have converged when the maximum difference between corresponding components of successive iterates is less than \( 5 \times 10^{-6} \)).

In a few cases the \( w(k) \) sequence does not converge, but continues to increase after reaching \( \omega_b \). This leads to an estimate of \( \bar{\omega} \) being calculated which is greater than unity. At this point, the process of estimating \( \omega_b \) is abandoned, and the most recent value used for the rest of the iterative solution of the equations. In these instances this most recent value is recorded as \( \omega_b(p) \) and the number of iterations performed when this stage is reached appears in parentheses in the \( n_{\omega_b} \) column.
Our results suggest that two main factors influence the increase in the number of iterations required when Kulsrud's technique is used instead of iterating with $\omega = \omega_b$ throughout. The first is the speed at which the $\omega^{(k)}$ sequence converges relative to the total number of iterations. The greater the proportion of iterations performed with $\omega = \omega_b$, the greater the overall rate of convergence ought to be. This is well illustrated by the strategy-1 results for Problem II, where the $\omega^{(k)}$ sequence converges early in the iteration and the increases are small (zero in two instances). The second factor is the accuracy with which $\omega^{(p)}$ estimates $\omega_b$. Although it has been shown by Young (1954) that using a value of slightly greater than the optimum does not slow down the rate of convergence of the SOR methods very much, some of the estimates $\omega^{(p)}$ obtained in our experiments are appreciable greater than $\omega_b$. Hence, of the four SOR methods, SPOR, in which the overestimation of $\omega_b$ is generally greatest, shows the greatest percentage increase in the number of iterations required.

We now consider the amount of work per point required for convergence in our experiments. In addition to the work associated with the SOR iteration, some effort is needed to compute the norms of the vectors from which $\lambda$ is estimated. This effort is comparatively small, but is included for the sake of completeness. It consists of two operations per point of the form

$$a + |b|$$

whenever a new estimate of $\omega_b$ is made. The modulus function is

'if $b < 0$ then subtract $b$ from zero'

which we consider as half a subtraction, on the assumption that $b$ is equally likely to be positive or negative.

Tables 4.14 and 4.15 show the work per point required by each method/strategy combination for Problems I and II respectively.
TABLE 4.12
Mildly non-linear equations, Problem I. SOR methods with Kulsrud's technique

<table>
<thead>
<tr>
<th>Method</th>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>$n_{wb}$</th>
<th>$\omega(p)$</th>
<th>Outer iterations</th>
<th>Inner iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point SOR 20</td>
<td>1</td>
<td>20</td>
<td>1.79</td>
<td>3</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>20</td>
<td>1.79</td>
<td>3</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>20</td>
<td>1.86</td>
<td>3</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>20</td>
<td>1.86</td>
<td>13</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>25</td>
<td>1.89</td>
<td>3</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>25</td>
<td>1.89</td>
<td>16</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>SLOR 20</td>
<td>1</td>
<td>15</td>
<td>1.70</td>
<td>3</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>15</td>
<td>1.70</td>
<td>7</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>20</td>
<td>1.80</td>
<td>3</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>20</td>
<td>1.80</td>
<td>10</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>20</td>
<td>1.85</td>
<td>3</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>20</td>
<td>1.85</td>
<td>12</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>S2LOR 20</td>
<td>1</td>
<td>15</td>
<td>1.58</td>
<td>3</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>15</td>
<td>1.58</td>
<td>5</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>15</td>
<td>1.70</td>
<td>3</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>15</td>
<td>1.70</td>
<td>7</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>20</td>
<td>1.77</td>
<td>3</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>20</td>
<td>1.77</td>
<td>9</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>SPOR 20</td>
<td>1</td>
<td>15</td>
<td>1.71</td>
<td>3</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>15</td>
<td>1.74</td>
<td>7</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>(30)</td>
<td>1.88</td>
<td>3</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>(30)</td>
<td>1.90</td>
<td>10</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>20</td>
<td>1.88</td>
<td>3</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>20</td>
<td>1.88</td>
<td>13</td>
<td>65</td>
<td></td>
</tr>
</tbody>
</table>
### TABLE 4.13
Mildly non-linear equations, Problem II. SOR methods with Kulsrud's technique

<table>
<thead>
<tr>
<th>Method</th>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>$n_{\omega_b}$</th>
<th>$\omega(p)$</th>
<th>Outer iterations</th>
<th>Inner iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point SOR</td>
<td>20</td>
<td>1</td>
<td>20</td>
<td>1.79</td>
<td>4</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>20</td>
<td>1.77</td>
<td>12</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>25</td>
<td>1.83</td>
<td>4</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>30</td>
<td>1.85</td>
<td>17</td>
<td>85</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>30</td>
<td>1.87</td>
<td>4</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>45</td>
<td>1.89</td>
<td>23</td>
<td>115</td>
</tr>
<tr>
<td>SLOR</td>
<td>20</td>
<td>1</td>
<td>15</td>
<td>1.66</td>
<td>4</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>15</td>
<td>1.66</td>
<td>9</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>1.79</td>
<td>4</td>
<td>93</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>20</td>
<td>1.76</td>
<td>12</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>25</td>
<td>1.82</td>
<td>4</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>25</td>
<td>1.82</td>
<td>15</td>
<td>75</td>
</tr>
<tr>
<td>S2LOR</td>
<td>20</td>
<td>1</td>
<td>15</td>
<td>1.55</td>
<td>4</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>15</td>
<td>1.55</td>
<td>7</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>1.68</td>
<td>4</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>20</td>
<td>1.68</td>
<td>9</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>25</td>
<td>1.75</td>
<td>4</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>25</td>
<td>1.75</td>
<td>12</td>
<td>60</td>
</tr>
<tr>
<td>SPOR</td>
<td>20</td>
<td>1</td>
<td>15</td>
<td>1.69</td>
<td>4</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>15</td>
<td>1.69</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>20</td>
<td>1.80</td>
<td>4</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>25</td>
<td>1.80</td>
<td>13</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>30</td>
<td>1.86</td>
<td>4</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>30</td>
<td>1.86</td>
<td>17</td>
<td>85</td>
</tr>
</tbody>
</table>
TABLE 4.14
Mildly non-linear equations, Problem I. Kulsrud's technique. Work per point

<table>
<thead>
<tr>
<th>Method</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Column 1: Method
- Column 2: Point SOR
- Column 3: SLOR
- Column 4: S2LOR
- Column 5: SPOR
- Values in the table are for different strategies and steps (h⁻¹)
<table>
<thead>
<tr>
<th>Method</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>1922</td>
<td>1212</td>
<td>1359</td>
<td>909</td>
</tr>
<tr>
<td>30</td>
<td>2605</td>
<td>1718</td>
<td>1902</td>
<td>1212</td>
</tr>
<tr>
<td>40</td>
<td>3248</td>
<td>2327</td>
<td>2325</td>
<td>1515</td>
</tr>
</tbody>
</table>

**TABLE 4.15**

Mildly non-linear equations, Problem II. Kulrsrud's technique. Work per point
Table 4.16 shows the various method/strategy combinations ranked in order of efficiency.

**TABLE 4.16**

*Mildly non-linear equations: Comparative efficiency of various methods and strategies using Kulsrud's technique*

<table>
<thead>
<tr>
<th>Efficiency ranking</th>
<th>Problem I Method</th>
<th>Strategy</th>
<th>Problem II Method</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>S2LOR</td>
<td>1</td>
<td>S2LOR</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>S2LOR</td>
<td>2</td>
<td>SLOR</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>SLOR</td>
<td>1</td>
<td>SPOR</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>SLOR</td>
<td>2</td>
<td>S2LOR</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Point SOR</td>
<td>1</td>
<td>Point SOR</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>SPOR</td>
<td>1</td>
<td>SLOR</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Point SOR</td>
<td>2</td>
<td>Point SOR</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>SPOR</td>
<td>2</td>
<td>SPOR</td>
<td>1</td>
</tr>
</tbody>
</table>
Although there are some differences between these results and those for the method/strategy combinations with $\omega = \omega_{b}$ throughout, the S2LOR method combined with strategy 1 for problem I and strategy 2 for problem II remains the most efficient approach.

4.5.2 Carré's Technique

If we write the SOR method in the form

$$\vec{z}(n+1) = M_{w} \vec{z}(n) + \vec{d}, \quad (4.26)$$

then using the argument and notation of sub-section 4.5.1 we have

$$\vec{z}(n+1) - \vec{z}(n) = \sum_{i=1}^{N} \alpha_{i} \lambda_{i}^{n} \vec{v}_{i}$$

$$= \lambda_{1}^{n} \left( \alpha_{1} \vec{v}_{1} + \sum_{i=2}^{N} \alpha_{i} \left( \frac{\lambda_{1}}{\lambda_{i}} \right)^{n} \vec{v}_{i} \right) \quad (4.27)$$

so that

$$||\vec{z}(n+1) - \vec{z}(n)|| \approx ||\alpha_{1}|| \left| \lambda_{1}^{n} \right| ||\vec{v}_{1}|| \quad (4.28)$$

The smaller the quantities $\left( \frac{\lambda_{1}}{\lambda_{i}} \right)^{n}$, $2\epsilon i \in N$, the more accurate the approximation (4.28) is. Therefore the quantity

$$\left| \frac{\lambda_{1}}{\lambda_{2}} \right| \quad (4.29)$$

where $\lambda_{2}$ is the eigenvalue of second largest modulus, is a measure of the accuracy of the approximation (4.28). Expression (4.29) is very close to unity when $\omega = 1$, and is unity when $\omega = \omega_{b}$, being maximised for some $\omega_{m}$ such that $1 < \omega_{m} < \omega_{b}$. The strategy proposed by Carré is to make an estimate $\omega_{m}^{(k)}$ of $\omega_{m}$, then iterate with this value to obtain an estimate $\lambda^{(k)}$ of $\lambda$. An approximation to the spectral radius of the Jacobi matrix is calculated by

$$\bar{\omega}^{(k)} = \frac{\lambda^{(k)} + \omega_{m}^{(k)} - 1}{\omega_{m}^{(k)} \lambda^{(k)}} \quad (4.30)$$
and an estimate of $\omega_b$ is made by

$$\omega^{(k)} = \frac{2}{1 + \sqrt{1 - \left\| \mathbf{u}^{(k)} \right\|^2}} \quad (4.31)$$

A new value for $\omega_m$ is then found by

$$\omega^{(k+1)}_m = \omega^{(k)} - \frac{2 - \omega^{(k)}_m}{4} \quad (4.32)$$

and the iteration proceeds using the current estimate of $\omega_m$ as $\omega$ until the $\omega^{(k)}$ sequence (NOT the $\omega^{(k)}_m$ sequence) converges to the desired accuracy. The final value of $\omega^{(k)}$ is then used for the remainder of the SOR iteration.

Experimental results are presented here for Carré's technique applied to Problems I and II using the combinations of SOR method and inner/outer iteration strategy studied earlier. Following Carré, we perform the first iteration with $\omega=1$, followed by a number of iterations with $\omega=1.375$ before starting to adaptively estimate $\omega_b$. We use two different numbers of iterations between making estimates of $\omega_b$: 5, which gives acceptable results with Kulsrud's technique, and 12, the number proposed by Carré. We call these version 1 and version 2 respectively.

For the former, the first outer iteration of strategy 2 methods is lengthened from 5 to 6 inner iterations, so that a new estimate of $\omega_b$ is made at the end of each outer iteration. For our first computed estimate of $\omega_b$ we use the $||.||_1$ norms of the four most recent displacement vectors to obtain three successive estimates of $\lambda$ using equation (4.28). If the differences between these estimates decrease and have the same sign, $\lambda^{(1)}$ is found by Aitken extrapolation, otherwise the last of the three estimates is taken as $\lambda^{(1)}$. Subsequent values of $\lambda^{(k)}$ are found using equation (4.18) only. We use the same test as for
Kulsrud's method to determine whether the $\omega^{(k)}$ sequence has converged, namely

$$\frac{|\omega^{(k+1)} - \omega^{(k)}|}{2 - \omega^{(k+1)}} < 0.05.$$ 

In the following tables, we record the number of inner iterations required for the $\omega^{(k)}$ sequence to converge ($n_{\omega_b}$), the converged value ($\omega^{(p)}$) and the number of outer and inner iterations required for the equations to converge. (The equations are deemed to have converged when the maximum difference between corresponding elements of successive iterates is less than $5 \times 10^{-6}$).

In a few cases, the $\omega^{(k)}$ sequence does not converge. Inspection of the $\omega^{(k)}$ sequences in these instances suggest that Carré's process is progressing satisfactorily but slowly, so that the equations converge before the $\omega^{(k)}$ sequence can. These cases are indicated in the tables by the values of $\omega^{(p)}$ (here the most recent value of $\omega^{(k)}$ obtained) and $n_{\omega_b}$ (the number of inner iterations required to reach this estimate) being in parentheses.
### TABLE 4.17

Mildly non-linear equations, Problem I. SOR methods with Carré's technique, Version I

<table>
<thead>
<tr>
<th>Method</th>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>$n_{\omega_0}$</th>
<th>$\omega(p)$</th>
<th>Outer iterations</th>
<th>Inner iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point SOR</td>
<td>20</td>
<td>1</td>
<td>16</td>
<td>1.76</td>
<td>3</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16</td>
<td>1.76</td>
<td>9</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>21</td>
<td>1.84</td>
<td>2</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>21</td>
<td>1.84</td>
<td>13</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>26</td>
<td>1.88</td>
<td>2</td>
<td>66</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>26</td>
<td>1.88</td>
<td>16</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>SLOR</td>
<td>20</td>
<td>1</td>
<td>16</td>
<td>1.67</td>
<td>3</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16</td>
<td>1.67</td>
<td>7</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>16</td>
<td>1.77</td>
<td>3</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16</td>
<td>1.77</td>
<td>9</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>21</td>
<td>1.83</td>
<td>2</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>21</td>
<td>1.83</td>
<td>11</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>S2LOR</td>
<td>20</td>
<td>1</td>
<td>11</td>
<td>1.55</td>
<td>3</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>11</td>
<td>1.55</td>
<td>5</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>16</td>
<td>1.68</td>
<td>2</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16</td>
<td>1.68</td>
<td>7</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>16</td>
<td>1.75</td>
<td>2</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16</td>
<td>1.75</td>
<td>9</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>SPOR</td>
<td>20</td>
<td>1</td>
<td>26</td>
<td>1.63</td>
<td>3</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>26</td>
<td>1.63</td>
<td>8</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>16</td>
<td>1.80</td>
<td>3</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>16</td>
<td>1.80</td>
<td>9</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>21</td>
<td>1.85</td>
<td>3</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>21</td>
<td>1.85</td>
<td>11</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>h⁻¹</td>
<td>Strategy</td>
<td>( n_{\omega_b} )</td>
<td>( \omega(p) )</td>
<td>Outer iterations</td>
<td>Inner iterations</td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>----------</td>
<td>------------------</td>
<td>----------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Point SOR</td>
<td>20</td>
<td>1</td>
<td>21</td>
<td>1.74</td>
<td>4</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>21</td>
<td>1.74</td>
<td>11</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>31</td>
<td>1.82</td>
<td>4</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>31</td>
<td>1.82</td>
<td>16</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>36</td>
<td>1.86</td>
<td>4</td>
<td>172</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>36</td>
<td>1.86</td>
<td>22</td>
<td>111</td>
</tr>
<tr>
<td>SLOR</td>
<td>20</td>
<td>1</td>
<td>21</td>
<td>1.65</td>
<td>4</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>21</td>
<td>1.65</td>
<td>9</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>26</td>
<td>1.75</td>
<td>4</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>26</td>
<td>1.75</td>
<td>13</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>31</td>
<td>1.80</td>
<td>4</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>31</td>
<td>1.80</td>
<td>16</td>
<td>81</td>
</tr>
<tr>
<td>S2LOR</td>
<td>20</td>
<td>1</td>
<td>11</td>
<td>1.53</td>
<td>4</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>11</td>
<td>1.53</td>
<td>6</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>21</td>
<td>1.66</td>
<td>4</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>21</td>
<td>1.66</td>
<td>9</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>26</td>
<td>1.73</td>
<td>4</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>26</td>
<td>1.73</td>
<td>12</td>
<td>61</td>
</tr>
<tr>
<td>SPOR</td>
<td>20</td>
<td>1</td>
<td>16</td>
<td>1.66</td>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>16</td>
<td>1.66</td>
<td>9</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>36</td>
<td>1.73</td>
<td>4</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>36</td>
<td>1.72</td>
<td>14</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>31</td>
<td>1.83</td>
<td>4</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>31</td>
<td>1.83</td>
<td>16</td>
<td>81</td>
</tr>
<tr>
<td>Method</td>
<td>$h^{-1}$</td>
<td>Strategy</td>
<td>$n_{_{o_a}}$</td>
<td>$\omega(p)$</td>
<td>Outer iterations</td>
<td>Inner iterations</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>--------------</td>
<td>-------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td><strong>Point SOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>(49)</td>
<td>(1.59)</td>
<td>2</td>
<td></td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>49</td>
<td>1.73</td>
<td>12</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>37</td>
<td>1.83</td>
<td>2</td>
<td></td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>37</td>
<td>1.83</td>
<td>15</td>
<td></td>
<td>75</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>49</td>
<td>1.87</td>
<td>2</td>
<td></td>
<td>76</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>49</td>
<td>1.87</td>
<td>18</td>
<td></td>
<td>90</td>
</tr>
<tr>
<td><strong>SLOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>25</td>
<td>1.65</td>
<td>3</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25</td>
<td>1.65</td>
<td>9</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>49</td>
<td>1.74</td>
<td>2</td>
<td></td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>49</td>
<td>1.74</td>
<td>13</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>37</td>
<td>1.82</td>
<td>2</td>
<td></td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>37</td>
<td>1.82</td>
<td>14</td>
<td></td>
<td>70</td>
</tr>
<tr>
<td><strong>S2LOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>25</td>
<td>1.53</td>
<td>3</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25</td>
<td>1.53</td>
<td>7</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>25</td>
<td>1.66</td>
<td>2</td>
<td></td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25</td>
<td>1.66</td>
<td>9</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>25</td>
<td>1.74</td>
<td>2</td>
<td></td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25</td>
<td>1.74</td>
<td>10</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td><strong>SPOR</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>25</td>
<td>1.64</td>
<td>3</td>
<td></td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25</td>
<td>1.64</td>
<td>9</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>(37)</td>
<td>(1.68)</td>
<td>2</td>
<td></td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>49</td>
<td>1.74</td>
<td>12</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>(49)</td>
<td>(1.74)</td>
<td>2</td>
<td></td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>61</td>
<td>1.79</td>
<td>15</td>
<td></td>
<td>65</td>
</tr>
</tbody>
</table>
# TABLE 4.20

Mildly non-linear equations, Problem II. SOR methods with Carré's technique, Version 2

<table>
<thead>
<tr>
<th>Method</th>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>$n_{\omega_b}$</th>
<th>$\omega(p)$</th>
<th>Outer iterations</th>
<th>Inner iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point SOR</td>
<td>20</td>
<td>1</td>
<td>37</td>
<td>1.73</td>
<td>4</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>37</td>
<td>1.73</td>
<td>14</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>49</td>
<td>1.82</td>
<td>4</td>
<td>148</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>49</td>
<td>1.82</td>
<td>19</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>61</td>
<td>1.86</td>
<td>4</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>61</td>
<td>1.85</td>
<td>26</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>SLOR</td>
<td>20</td>
<td>1</td>
<td>25</td>
<td>1.65</td>
<td>4</td>
<td>77</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>37</td>
<td>1.64</td>
<td>12</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>37</td>
<td>1.75</td>
<td>4</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>37</td>
<td>1.75</td>
<td>14</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>49</td>
<td>1.81</td>
<td>4</td>
<td>139</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>49</td>
<td>1.81</td>
<td>18</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>S2LOR</td>
<td>20</td>
<td>1</td>
<td>25</td>
<td>1.53</td>
<td>4</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25</td>
<td>1.53</td>
<td>8</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>25</td>
<td>1.66</td>
<td>4</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>37</td>
<td>1.87</td>
<td>19</td>
<td>95 (1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>37</td>
<td>1.73</td>
<td>4</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>61</td>
<td>1.72</td>
<td>16</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>SPOR</td>
<td>20</td>
<td>1</td>
<td>25</td>
<td>1.64</td>
<td>4</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>25</td>
<td>1.64</td>
<td>10</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>49</td>
<td>1.74</td>
<td>4</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>37</td>
<td>1.75</td>
<td>14</td>
<td>70</td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>37</td>
<td>1.81</td>
<td>4</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>49</td>
<td>1.80</td>
<td>18</td>
<td>90</td>
<td></td>
</tr>
</tbody>
</table>

(1) Estimation of $\omega_b$ abandoned when $\mu^2$ becomes greater than unity
We note that the estimates \( w^{(p)} \) of \( \omega_b \) produced by Carré's technique are not always as good as those produced using the method of Kulsrud. In a substantial number of the experiments in the foregoing tables, \( w^{(p)} \) underestimates \( \omega_b \), which, as stated previously, is undesirable. However, this does not seem to affect the convergence of the solution adversely. The results obtained for the number of iterations required in the version 1 examples are not significantly different from those obtained for Kulsrud's method. Version 2 examples, however, require a few more iterations in almost every instance. This would appear to be because a greater proportion of the iterations are performed with a value of \( \omega \) much less than the optimum than in version 1. The fact that in strategy 2 methods the calculations of values of \( \omega_m^{(k)} \) are aligned with the end of outer iterations in version 1 but not in version 2 seems to be irrelevant.

We now consider the amount of work per point required for convergence in our experiments. In addition to the work associated with the basic solution process, there is a small amount of work each time a value of \( \omega_m^{(k)} \) is calculated. This consists of four operations per point of the form

\[
 a + |b| 
\]

(4.33)

for \( k=1 \) and two operations per point of this form for \( k>1 \). Since the vast majority of version 1 experiments require less iterations than the corresponding version 2 experiments, we consider only the former. Tables 4.20 and 4.21 give the work per point for problems I and II respectively.
### TABLE 4.21

Mildly non-linear equations, Problem I. Carré's technique, Version 1

Work per point

<table>
<thead>
<tr>
<th>Method</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>899.5</td>
<td>1065.5</td>
<td>779.5</td>
<td>835.5</td>
</tr>
<tr>
<td>30</td>
<td>1100</td>
<td>1528.5</td>
<td>939.5</td>
<td>1065.5</td>
</tr>
<tr>
<td>40</td>
<td>1383</td>
<td>1876.5</td>
<td>1040</td>
<td>1298.5</td>
</tr>
</tbody>
</table>
**TABLE 4.22**

Mildly non-linear equations, Problem II. Carré's technique, Version 1

Work per point

<table>
<thead>
<tr>
<th>Method</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>1985</td>
<td>1133.5</td>
<td>1505</td>
<td>933.5</td>
</tr>
<tr>
<td>30</td>
<td>2751</td>
<td>1639.5</td>
<td>2068</td>
<td>1336.5</td>
</tr>
<tr>
<td>40</td>
<td>3494</td>
<td>2242.5</td>
<td>2651</td>
<td>1639.5</td>
</tr>
</tbody>
</table>
Table 4.23 shows the various method/strategy combinations ranked in order of efficiency.

**TABLE 4.23**

Mildly non-linear equations. Comparative efficiency of various methods and strategies using Carré's technique

<table>
<thead>
<tr>
<th>Efficiency ranking</th>
<th>Problem I</th>
<th>Problem II</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Method</td>
<td>Strategy</td>
</tr>
<tr>
<td>1</td>
<td>S2LOR</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>SLOR</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>S2LOR</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>SLOR</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>Point SOR</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>SPOR</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>Point SOR</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>SPOR</td>
<td>2</td>
</tr>
</tbody>
</table>

There are some differences between these results and those for the method/strategy combinations with $\omega=\omega_b$ throughout, but the S2LOR method combined with strategy 1 for problem I and strategy 2 for problem II are still the best combinations.
4.5.3 Reid's Technique

We can write the coefficient matrix $A$ of the finite difference equations in the form

$$A = D - L - U$$

(4.34)

where $D$ is a diagonal matrix and $L$ and $U$ are the negatives of the strictly lower triangular and strictly upper triangular parts of $A$ respectively. If $A$ has property (A), and is consistently ordered, then for any positive scalar $r$, there exists a diagonal matrix $G_r$ such that

$$r^\frac{1}{2}L + r^{-\frac{1}{2}}U = G_r (L+U) G_r^{-1} \cdot$$

(4.35)

A matrix $G_r$ which satisfies this equation is defined as follows. Let $A = (a_{i,j})$ be of order $N$, and let $(s_1, s_2, \ldots, s_N)$ be an ordering vector of $A$ with integer coefficients as defined in Chapter 1. Since $A$ is consistently ordered, then for every non-zero $a_{i,j}$ such that $i \neq j$, either $i > j$ and $s_i = s_j + 1$, or $i < j$ and $s_i = s_j - 1$. The matrix $G_r$ is then

$$G_r = \text{diag} \left\{ r^{s_i/2} \right\} .$$

(4.36)

The SOR iteration matrix $M_\omega$ is defined as

$$M_\omega = (D - \omega L)^{-1} \left( (1 - \omega) D + \omega U \right) .$$

(4.37)

If this has an eigenvalue $\lambda_1$, the corresponding eigenvector $\chi_1$ satisfies the equation

$$((1 - \omega) D + \omega U) \chi_1 = \lambda_1 (D - \omega L) \chi_1 \cdot$$

(4.38)

which can be rewritten as

$$\omega \lambda_1^\frac{1}{2} (\lambda_1^\frac{1}{2} L - \lambda_1^{-\frac{1}{2}} U) \chi_1 = (\lambda_1 + \omega - 1) D \chi_1 .$$

(4.38')

Making use of equation (4.35) gives:

$$D^{-\frac{1}{2}} (L+U) G^{-1}_\lambda \chi_1 = \frac{\lambda_1 + \omega - 1}{\omega \lambda_1^\frac{1}{2}} \frac{G^{-1}_\lambda}{\chi_1} \cdot$$

(4.39)

Now $D^{-\frac{1}{2}} (L+U)$ is the Jacobi iteration matrix, and from equation (4.39)
we see that it has eigenvectors
\[ x_i = G^{-1} \lambda_i y_i \]  (4.40)
and corresponding eigenvalues
\[ \mu_i = \frac{\lambda_i\omega - 1}{\omega\lambda_i^4} . \]  (4.41)

The SOR iterative process has the form
\[ z^{(n+1)} = M z^{(n)} + d , \quad n=0,1,2,\ldots , \]  (4.42)
with \( z^{(0)} \) arbitrary. Reid's technique is to use the ratio of the norms of successive displacement vectors
\[ \frac{||z^{(n+1)} - z^{(n)}||_2}{||z^{(n)} - z^{(n-1)}||_2} \]  (4.43)
to give an estimate of \( \bar{\lambda} \), the spectral radius of \( M \), and the displacement vector itself, \( z^{(n+1)} - z^{(n)} \), to give approximation to \( y_1 \), the corresponding eigenvector (the dominant eigenvector). An estimate of \( x_1 \), the dominant eigenvector of the Jacobi matrix, can then be found from \( y_1 \) using equation (4.40). An estimate of the spectral radius of the Jacobi matrix can be made from \( x_1 \) using a Rayleigh quotient procedure, i.e.
\[ \mu = \frac{x_1^T(L+U)x_1}{x_1^TDx_1} . \]  (4.44)

From this, a new estimate of the optimum \( \omega \) can be made using the usual formula, i.e.
\[ \omega_b = \frac{2}{1+\sqrt{1-\bar{\mu}^2}} . \]  (4.45)

The above argument strictly applies only to the point SOR method. It can be adapted for the block methods as follows. In such methods, the matrix \( A \) has the form,
If we let

$$A = \begin{bmatrix}
A_{1,1} & A_{1,2} & & & & & \\
A_{2,1} & A_{2,2} & A_{2,3} & & & & \\
& A_{3,2} & A_{3,3} & A_{3,4} & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
\end{bmatrix}$$

$$D = \begin{bmatrix}
A_{1,1} & & & & & \\
& A_{2,2} & A_{3,3} & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
& & & & & \\
\end{bmatrix}$$

$$L = \begin{bmatrix}
0 & -A_{2,1} & 0 & & & \\
0 & -A_{3,2} & & & & \\
0 & 0 & & & & \\
0 & 0 & -A_{m-1,m-2} & 0 & \\
0 & 0 & 0 & -A_{m,m} & \\
\end{bmatrix}$$

and

$$U = \begin{bmatrix}
0 & -A_{1,2} & & & & \\
0 & 0 & -A_{2,3} & & & \\
0 & 0 & -A_{3,4} & & & \\
0 & 0 & 0 & -A_{m-1,m} & & \\
0 & 0 & 0 & 0 & \end{bmatrix}$$

(4.46)
we can construct a matrix $G_r$ which satisfies equation (4.35) as follows. Let $(t_1, t_2, \ldots, t_m)$ be an ordering vector for the matrix $A$ in equation (4.46). Then, since $A$ is block consistently ordered, for every sub-matrix $A_{i,j}$ which is not null, either $i>j$ and $t_i = t_j + 1$, or $i<j$ and $t_i = t_j - 1$. If we let $I_{j,j}$ represent the identity matrix of the same order as $A_{j,j}$, then

$$G_r = \begin{bmatrix}
  t_1/2 & I_{1,1} & & & \\
  & t_2/2 & I_{2,2} & & \\
  & & t_3/2 & I_{3,3} & \\
  & & & \ddots & \ddots \\
  & & & & t_m/2 I_{m,m}
\end{bmatrix}$$

satisfies equation (4.35) and the rest of the argument is identical to that for the point SOR method.

Reid's technique is to perform two SOR iterations with $\omega = \omega^{(1)} = 1$, then calculate $\lambda^{(1)}$, an estimate of $\lambda$, using expression (4.43). A new value of $\omega, \omega^{(2)}$ is then found as described above. (We note that the estimate of $\bar{\mu}$ cannot exceed its true value, since $\bar{\mu}$ is defined as

$$\bar{\mu} = \max \frac{x^T(L+U)x}{x^TDx} .$$

Hence, the values $\omega^{(1)}, \omega^{(2)}, \ldots$ cannot exceed $\omega_b$). New estimates of $\omega_b$ are then made at regular intervals until the $\omega^{(k)}$ sequence converges. The sequence is deemed to have converged when

$$(\lambda^{(k)})^6 < (\omega^{(k)} - 1)^5 .$$

The $\omega^{(k)}$ sequence is monotonically increasing at first, but rounding error may cause it to decrease in the latter stages of the iteration.
In cases where $w^{(k+1)}$ is less than $w^{(k)}$, the older value is retained.

Experimental results are presented here for Reid's method applied to the solution of problems I and II using the combinations of SOR method and inner/outer iteration strategy as before. We use two different numbers of iterations between making estimates of $w_b$: 5, which gives acceptable results with Kulsrud's method, and 12, the number recommended by Reid. We call these version 1 and version 2 respectively. For the former, the first order iteration of strategy 2 methods is lengthened from 5 to 7 inner iterations, so that a new estimate of $w_b$ is made at the end of each outer iteration.

The $w^{(k)}$ sequence failed to converge in every one of the experiments recorded here. In the majority of instances, one or more acceptable estimates of $w_b$ were found, then all subsequent estimates were rejected as being less than the existing one. Exceptionally, significant improvements in the estimate of $w_b$ were made after one or two values had been rejected. On this basis, the estimating process is terminated when three values of $w^{(k)}$ have been rejected as being too small.

In the following tables, we record the best (i.e. largest) value of $w^{(k)}$ obtained ($w^{(p)}$), the number of inner iterations required to find this value ($n_w$), and the number of outer and inner iterations required for the equations to converge. (The equations are deemed to have converged when the maximum difference between corresponding elements of successive iterates is less than $5\times10^{-6}$).
## TABLE 4.24

Mildly non-linear equations, Problem I. SOR methods with Reid's technique, Version 1

<table>
<thead>
<tr>
<th>Method</th>
<th>( h^{-1} )</th>
<th>Strategy</th>
<th>( n_{\omega_b} )</th>
<th>( \omega(p) )</th>
<th>Outer iterations</th>
<th>Inner iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point SOR</td>
<td>20</td>
<td>1</td>
<td>17</td>
<td>1.72</td>
<td>2</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>17</td>
<td>1.72</td>
<td>9</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>22</td>
<td>1.80</td>
<td>2</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>22</td>
<td>1.80</td>
<td>13</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>32</td>
<td>1.85</td>
<td>2</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>32</td>
<td>1.85</td>
<td>16</td>
<td>82</td>
</tr>
<tr>
<td>SLOR</td>
<td>20</td>
<td>1</td>
<td>12</td>
<td>1.63</td>
<td>3</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>12</td>
<td>1.63</td>
<td>7</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>17</td>
<td>1.73</td>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>17</td>
<td>1.73</td>
<td>10</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>22</td>
<td>1.79</td>
<td>2</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>22</td>
<td>1.79</td>
<td>13</td>
<td>67</td>
</tr>
<tr>
<td>S2LOR</td>
<td>20</td>
<td>1</td>
<td>7</td>
<td>1.51</td>
<td>3</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>7</td>
<td>1.51</td>
<td>5</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>17</td>
<td>1.64</td>
<td>3</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>17</td>
<td>1.64</td>
<td>8</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>22</td>
<td>1.72</td>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>22</td>
<td>1.72</td>
<td>10</td>
<td>52</td>
</tr>
<tr>
<td>SPOR</td>
<td>20</td>
<td>1</td>
<td>22</td>
<td>1.63</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>22</td>
<td>1.63</td>
<td>6</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>22</td>
<td>1.74</td>
<td>3</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>32</td>
<td>1.74</td>
<td>9</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>42</td>
<td>1.79</td>
<td>3</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>47</td>
<td>1.79</td>
<td>11</td>
<td>57</td>
</tr>
</tbody>
</table>
TABLE 4.25

Mildly non-linear equations, Problem II. SOR methods with Reid's technique, Version 1

<table>
<thead>
<tr>
<th>Method</th>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>$n_{\omega_b}$</th>
<th>$\omega(p)$</th>
<th>Outer iterations</th>
<th>Inner iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point SOR</td>
<td>20</td>
<td>1</td>
<td>22</td>
<td>1.72</td>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>22</td>
<td>1.72</td>
<td>12</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>32</td>
<td>1.81</td>
<td>4</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>27</td>
<td>1.81</td>
<td>17</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>37</td>
<td>1.85</td>
<td>4</td>
<td>177</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>37</td>
<td>1.85</td>
<td>22</td>
<td>112</td>
</tr>
<tr>
<td>SLOR</td>
<td>20</td>
<td>1</td>
<td>17</td>
<td>1.62</td>
<td>4</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>12</td>
<td>1.63</td>
<td>9</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>22</td>
<td>1.73</td>
<td>4</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>22</td>
<td>1.73</td>
<td>13</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>27</td>
<td>1.80</td>
<td>4</td>
<td>129</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>27</td>
<td>1.80</td>
<td>16</td>
<td>82</td>
</tr>
<tr>
<td>S2LOR</td>
<td>20</td>
<td>1</td>
<td>12</td>
<td>1.52</td>
<td>4</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>12</td>
<td>1.52</td>
<td>6</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>17</td>
<td>1.65</td>
<td>4</td>
<td>79</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>17</td>
<td>1.65</td>
<td>9</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>22</td>
<td>1.72</td>
<td>4</td>
<td>99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>22</td>
<td>1.72</td>
<td>12</td>
<td>62</td>
</tr>
<tr>
<td>SPOR</td>
<td>20</td>
<td>1</td>
<td>17</td>
<td>1.64</td>
<td>4</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>17</td>
<td>1.64</td>
<td>10</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>22</td>
<td>1.74</td>
<td>4</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>22</td>
<td>1.74</td>
<td>14</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>32</td>
<td>1.80</td>
<td>4</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>32</td>
<td>1.79</td>
<td>18</td>
<td>92</td>
</tr>
</tbody>
</table>
**TABLE 4.26**


<table>
<thead>
<tr>
<th>Method</th>
<th>( h^{-1} )</th>
<th>Strategy</th>
<th>( n_{\omega_b} )</th>
<th>( \omega(p) )</th>
<th>Outer iterations</th>
<th>Inner iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point SOR</td>
<td>20</td>
<td>1</td>
<td>14</td>
<td>1.73</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.73</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>14</td>
<td>1.80</td>
<td>3</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.80</td>
<td>14</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>38</td>
<td>1.85</td>
<td>3</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>74</td>
<td>1.85</td>
<td>18</td>
<td>90</td>
</tr>
<tr>
<td>SLOR</td>
<td>20</td>
<td>1</td>
<td>14</td>
<td>1.63</td>
<td>3</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.63</td>
<td>8</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>14</td>
<td>1.74</td>
<td>3</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.74</td>
<td>11</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>14</td>
<td>1.79</td>
<td>3</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>62</td>
<td>1.79</td>
<td>14</td>
<td>70</td>
</tr>
<tr>
<td>S2LOR</td>
<td>20</td>
<td>1</td>
<td>14</td>
<td>1.52</td>
<td>3</td>
<td>29</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.52</td>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>14</td>
<td>1.65</td>
<td>2</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.65</td>
<td>9</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>38</td>
<td>1.72</td>
<td>3</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>38</td>
<td>1.72</td>
<td>11</td>
<td>55</td>
</tr>
<tr>
<td>SPOR</td>
<td>20</td>
<td>1</td>
<td>14</td>
<td>1.64</td>
<td>3</td>
<td>28</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.64</td>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>14</td>
<td>1.74</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.74</td>
<td>9</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>14</td>
<td>1.80</td>
<td>3</td>
<td>47</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.80</td>
<td>11</td>
<td>55</td>
</tr>
</tbody>
</table>
TABLE 4.27
Mildly non-linear equations, Problem II. SOR methods with Reid's technique, Version 2

<table>
<thead>
<tr>
<th>Method</th>
<th>h⁻¹</th>
<th>Strategy</th>
<th>n_{ω_b}</th>
<th>ω(p)</th>
<th>Outer iterations</th>
<th>Inner iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point SOR</td>
<td>20</td>
<td>1</td>
<td>26</td>
<td>1.72</td>
<td>4</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>26</td>
<td>1.72</td>
<td>15</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>38</td>
<td>1.81</td>
<td>4</td>
<td>150</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>38</td>
<td>1.81</td>
<td>21</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>50</td>
<td>1.85</td>
<td>4</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>50</td>
<td>1.85</td>
<td>25</td>
<td>125</td>
</tr>
<tr>
<td>SLOR</td>
<td>20</td>
<td>1</td>
<td>14</td>
<td>1.63</td>
<td>5</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.63</td>
<td>11</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>26</td>
<td>1.74</td>
<td>4</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>26</td>
<td>1.73</td>
<td>16</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>26</td>
<td>1.79</td>
<td>4</td>
<td>138</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>26</td>
<td>1.79</td>
<td>18</td>
<td>90</td>
</tr>
<tr>
<td>S2LOR</td>
<td>20</td>
<td>1</td>
<td>14</td>
<td>1.52</td>
<td>5</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>14</td>
<td>1.52</td>
<td>9</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>26</td>
<td>1.64</td>
<td>5</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>26</td>
<td>1.64</td>
<td>12</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>26</td>
<td>1.72</td>
<td>4</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>26</td>
<td>1.72</td>
<td>14</td>
<td>70</td>
</tr>
<tr>
<td>SPOR</td>
<td>20</td>
<td>1</td>
<td>14</td>
<td>1.64</td>
<td>4</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>26</td>
<td>1.64</td>
<td>11</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>1</td>
<td>26</td>
<td>1.74</td>
<td>4</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>26</td>
<td>1.74</td>
<td>15</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>1</td>
<td>62</td>
<td>1.79</td>
<td>4</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>74</td>
<td>1.79</td>
<td>19</td>
<td>95</td>
</tr>
</tbody>
</table>
As expected, the values $w(p)$ never exceed the corresponding optimum values of $w$, although they approach closely. The number of inner iterations required do not differ much overall from those required by the techniques of Kulsrud and Carré, although, as observed with Carré's technique, version 2 leads to slightly more inner iterations than version 1. In a few cases, estimates of $\lambda$ were obtained which were greater than unity. This only occurred when the estimate was made early in an outer iteration. As it is well known that iterative processes behave erratically at the beginning, this is not a surprising result. In these cases, the previous estimate of $w_b$ was retained.

We now consider the amount of work per point required for convergence. In addition to the work associated with the basic solution process, there is a non-trivial amount of work performed each time a new value of $w(k)$ is calculated. The norms of the displacement vectors are found by two operations per point of the form

$$a + b \times b,$$

which gives a total of 7 in our weighting scheme. If we store a vector whose elements are a half of those of the ordering vector for the coefficient matrix, then the evaluation of the matrix $G^{-1}_1$ requires one operation per point of the form

$$a^b$$

in the point SOR method, but only one operation per block of this form in the block methods. Since the number of blocks is small compared with the number of points, we shall only take account of this work in the point SOR method. Assuming the computer implementation of this operation to be

$$\exp(b \times \log(a)),$$

we estimate its execution time as 30. If the coefficient matrix $A$ has
its diagonal elements equal to 4 and its non-zero off-diagonal elements equal to -1, the evaluation of the Rayleigh quotient (4.44) can be achieved with one multiplication and four additions per point which is an execution time of 6.5.

Although the version 2 experiments almost invariably require a few more inner iterations than the corresponding version 1 experiments, they also need less applications of Reid's procedure, and hence it is not clear whether version 1 or version 2 is more efficient. The results for both versions are therefore presented in the following tables.
<table>
<thead>
<tr>
<th>Method</th>
<th>Point SOR</th>
<th>SLO</th>
<th>S2LO</th>
<th>SPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strategy</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>1060.5</td>
<td>1327.5</td>
<td>808.5</td>
<td>923</td>
</tr>
<tr>
<td>30</td>
<td>1397</td>
<td>1835.5</td>
<td>959.5</td>
<td>1281.5</td>
</tr>
<tr>
<td>40</td>
<td>1750</td>
<td>2242</td>
<td>1193</td>
<td>1640</td>
</tr>
</tbody>
</table>
TABLE 4.29

Mildly non-linear equations, problem II. Reid's technique, version 1.

Work per point

<table>
<thead>
<tr>
<th>Method</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>h⁻¹</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>2332</td>
<td>1529</td>
<td>1624.5</td>
<td>1018</td>
</tr>
<tr>
<td>30</td>
<td>3215</td>
<td>2065.5</td>
<td>2198</td>
<td>1445</td>
</tr>
<tr>
<td>40</td>
<td>3971.5</td>
<td>2638.5</td>
<td>2731.5</td>
<td>1758.5</td>
</tr>
</tbody>
</table>
TABLE 4.30

Mildly non-linear equations, problem I. Reid's technique, version 2.

Work per point

<table>
<thead>
<tr>
<th>Method</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>991</td>
<td>1332.5</td>
<td>808</td>
<td>974</td>
</tr>
<tr>
<td>30</td>
<td>1410</td>
<td>1792.5</td>
<td>1041.5</td>
<td>1332.5</td>
</tr>
<tr>
<td>40</td>
<td>1735</td>
<td>2362</td>
<td>1275</td>
<td>1731.5</td>
</tr>
</tbody>
</table>
TABLE 4.3:1

Mildly non-linear equations, problem II. Reid's technique, version 2.

Work per point

<table>
<thead>
<tr>
<th>Method</th>
<th>Point SOR</th>
<th>SLOR</th>
<th>S2LOR</th>
<th>SPOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h⁻¹</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

|        | 2409      | 1719 | 1725  | 1167.5 |
|        | 3285.5    | 2355.5 | 2271 | 1681 |
|        | 4042      | 2792 | 2871  | 1881 |

|        | 1585.5    | 1167.5 | 1111.5 | 2095.5 |
|        | 2213.5    | 1477.5 | 2946  | 2106 |
|        | 2568.5    | 1726  | 3823.5 | 2673 |

199
The results for the two versions do not differ a great deal, but version 1 is generally the more efficient. Table 4.32 shows the various method/strategy combinations ranked in order of efficiency.

<table>
<thead>
<tr>
<th>Efficiency ranking</th>
<th>Problem I</th>
<th>Problem II</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Method</td>
<td>Strategy</td>
</tr>
<tr>
<td>1</td>
<td>S2LOR</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>SLOR</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>S2LOR</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>SPOR</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SLOR</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>Point</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>SOR</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>SPOR</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>Point</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>SOR</td>
<td></td>
</tr>
</tbody>
</table>

Although there are some differences between these results and those for the method/strategy combinations with \( \omega = a \) throughout, the S2LOR method combined with strategy 1 for problem I and strategy 2 for problem II are still the best combinations.
4.5.4 Conclusions

Our experiments support Reid's statement that his method is more reliable than those of Kulsrud or Carre. However, it does require a little more computational effort to solve the finite-difference equations and it also needs more storage. A bigger drawback is that it is more complicated than the other two methods and hence is likely to need more programmer and computer time to produce a working program.

On balance, Kulsrud's technique is probably the best of the three, as it needs the least computational effort and is the simplest to program. Our experiments indicate that when the method 'fails', the solution of the finite-difference equations may nevertheless be found with reasonable effort.
4.6 **PRECONDITIONED METHODS OF SIMULTANEOUS ITERATION**

We now consider the application of preconditioned iterative methods to the solution of mildly non-linear equations. It has already been shown (Chapter 2) that the finite-difference representations of equations (4.1) and (4.2) have coefficient matrices which are positive definite, and so preconditioned iterative methods described earlier can be used to solve them.

The preconditioned form of the finite-difference equations (4.3) can be written as:

\[
(I-wL)^{-1}A(I-wU)^{-1}[(I-wU)z] = (I-wL)^{-1}f(z). \tag{4.52}
\]

If the iterative process is carried out using the transformed vector 
\((I-wU)z\), it is necessary to evaluate \(z\) explicitly at the beginning of each outer iteration, in order to find the new value of \(f(z)\), which must then be premultiplied by \((I-wL)^{-1}\). This complication can be avoided and computational effort saved by using the alternative form of preconditioning,

\[
(I-wU)^{-1}(I-wL)^{-1}Az = (I-wU)^{-1}(I-wL)^{-1}f(z). \tag{4.52'}
\]

An inner iteration of the preconditioned simultaneous displacement is then written (using the notation of Figure 4.1) as

\[
z^{(i,j+1)} = z^{(i,j)} + \alpha(I-wU)^{-1}(I-wL)^{-1}(b-Az^{(i,j)}), \tag{4.53}
\]

where \(b=f(z^{(i-1,j)})\) is evaluated at the start of the current outer iteration.

The following table shows the maximum and minimum eigenvalues and P-condition numbers of the coefficient matrix \(A\) with and without preconditioning for various mesh sizes. The optimum values of the preconditioning parameter \(\omega\) are determined within an accuracy of ±0.05. The notation used is that of Chapter 2.
**TABLE 4.33**

Mildly non-linear equations. Eigenvalues of coefficient matrices with
and without preconditioning

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Without pre-conditioning</th>
<th>With pre-conditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\lambda_1$</td>
<td>$\lambda_N$</td>
</tr>
<tr>
<td>20</td>
<td>1.987690</td>
<td>0.012314</td>
</tr>
<tr>
<td>30</td>
<td>1.994529</td>
<td>0.005483</td>
</tr>
<tr>
<td>40</td>
<td>1.996925</td>
<td>0.003091</td>
</tr>
</tbody>
</table>
Table 4.33 indicates that, for this problem the preconditioning process reduces the P-condition number of the coefficient matrix so that

\[ P(\omega_b) \approx 0.5 \sqrt{P(0)} \]

which is a very good result. The optimum parameters for the simultaneous displacement and second-order Richardson's method are as follows.
TABLE 4.34
Mildly non-linear equations. Optimum parameters of the simultaneous displacement and second-order Richardson method

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simultaneous displacement</td>
<td>Second-order Richardson</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>20</td>
<td>1.0000</td>
<td>1.7294</td>
</tr>
<tr>
<td>30</td>
<td>1.0000</td>
<td>1.8106</td>
</tr>
<tr>
<td>40</td>
<td>1.0000</td>
<td>1.8543</td>
</tr>
</tbody>
</table>
Mildly non-linear equations, problem I. Experimental results for the simultaneous displacement and second-order Richardson methods

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>Without preconditioning</th>
<th></th>
<th>With preconditioning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simultaneous displacement</td>
<td>Second-order Richardson</td>
<td>Simultaneous displacement</td>
<td>Second-order Richardson</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Outer iterations</td>
<td>Inner iterations</td>
<td>Outer iterations</td>
<td>Inner iterations</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>2</td>
<td>245</td>
<td>2</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>75</td>
<td>375</td>
<td>13</td>
<td>65</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>2</td>
<td>402</td>
<td>2</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>139</td>
<td>695</td>
<td>18</td>
<td>90</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>2</td>
<td>527</td>
<td>2</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>209</td>
<td>1045</td>
<td>25</td>
<td>125</td>
</tr>
</tbody>
</table>
TABLE 4.36

Mildly non-linear equations, problem II. Experimental results for the simultaneous displacement and second-order Richardson methods

<table>
<thead>
<tr>
<th>h(^{-1})</th>
<th>Strategy</th>
<th>Without pre-conditioning</th>
<th>With pre-conditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simultaneous displacement</td>
<td>Second-order Richardson</td>
</tr>
<tr>
<td></td>
<td>Outer iterations</td>
<td>Inner iterations</td>
<td>Outer iterations</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>4</td>
<td>805</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>135</td>
<td>675</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>3</td>
<td>1494</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>275</td>
<td>1375</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>3</td>
<td>2266</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>452</td>
<td>2260</td>
</tr>
</tbody>
</table>
The question arises as to whether the preconditioning technique can be used effectively in conjunction with the linearised method of solution, where the coefficient matrix changes every outer iteration. To obtain the fastest rate of convergence, it is strictly necessary to re-calculate the optimum preconditioning parameter and the parameter(s) of the iterative method for each outer iteration. This is impractical because of the amount of computation required. However, provided the changes in the coefficient matrix during the solution process are small (so that diagonal dominance is maintained), the parameters derived from the original coefficient matrix produce a rate of convergence very close to the optimum. For the changes to be small, we require in this case that (using the notation of (4.4))

\[ |k^2 z_{i,j}| \lesssim 4, \quad i=1,2,...,n, \quad j=1,2,...,n \]

Throughout the process. If the order of magnitude of the solution is known, this condition can be assured by suitable choice of \( n \).

Table 4.37 shows the results of applying the linearised method of solution to Problem II, using the same parameters throughout the iteration.
Mildly non-linear equations, problem II (linearised). Experimental results for the simultaneous displacement and second-order Richardson methods

<table>
<thead>
<tr>
<th>h^{-1}</th>
<th>Strategy</th>
<th>Without preconditioning</th>
<th></th>
<th>With preconditioning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simultaneous displacement</td>
<td></td>
<td>Simultaneous displacement</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Outer iterations</td>
<td>Inner iterations</td>
<td>Outer iterations</td>
<td>Inner iterations</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>3</td>
<td>796</td>
<td>4</td>
<td>142</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>135</td>
<td>675</td>
<td>17</td>
<td>85</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>4</td>
<td>1488</td>
<td>3</td>
<td>182</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>275</td>
<td>1375</td>
<td>25</td>
<td>125</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>3</td>
<td>2259</td>
<td>3</td>
<td>261</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>452</td>
<td>2260</td>
<td>39</td>
<td>195</td>
</tr>
</tbody>
</table>
The foregoing tables show the very substantial gains in the rate of convergence which can be achieved using the preconditioning technique. The execution times per outer iteration, determined using our weighting scheme are as follows.

**TABLE 4.38**

Mildly non-linear equations with preconditioning. Execution times for outer iterations

<table>
<thead>
<tr>
<th></th>
<th>Problem I</th>
<th>Problem II</th>
<th>Problem II (linearised)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>22.5</td>
<td>7.5</td>
<td>9.5</td>
</tr>
</tbody>
</table>

The execution times per inner iteration depend on the iterative method, the strategy, and whether or not preconditioning is used. These times are shown in Table 4.39.

**TABLE 4.39**

Mildly non-linear equations with preconditioning. Execution times for inner iterations (methods without linearisation)

<table>
<thead>
<tr>
<th>Method</th>
<th>Strategy</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simultaneous</td>
<td>1</td>
<td>20</td>
<td>34</td>
</tr>
<tr>
<td>displacement</td>
<td>2</td>
<td>18.5</td>
<td>32.5</td>
</tr>
<tr>
<td>Second-order</td>
<td>1</td>
<td>23.5</td>
<td>37.5</td>
</tr>
<tr>
<td>Richardson</td>
<td>2</td>
<td>22</td>
<td>36</td>
</tr>
</tbody>
</table>
The corresponding figures for the linearised problem are 1.0 less in each case, since there is no right-hand side to be added to the iterate.

With these figures, we can estimate the computational effort required by the various approaches to solving the non-linear equations.

Tables 4.40-4.42 show clearly the savings on computational effort that can be made by applying preconditioning to methods of simultaneous iteration. The second-order Richardson method requires less work than the simultaneous displacement method, though at the expense of needing extra storage. The linearised version of problem II requires a little less work than the version without linearisation, but the saving is trivial, being about 2\% for the most efficient approach, the preconditioned second-order Richardson method, strategy 2.

As with the SOR methods, strategy 1 gives the better results for problem I, while strategy 2 is better for problem II. The results differ from those for the SOR methods in that here, the difference between the strategies tends to decrease as $h^{-1}$ increases.

Although the preconditioning technique is very powerful, our experiments indicate that the best results obtained using it are not quite as good as the best results obtained using block SOR. On the other hand, preconditioned methods of simultaneous iteration are rather simpler to program than block SOR methods.
### TABLE 4.40

Mildly non-linear equations, problem I. Work per point

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>Without preconditioning</th>
<th></th>
<th>With preconditioning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simultaneous displacement</td>
<td>Second-order Richardson</td>
<td>Simultaneous displacement</td>
<td>Second-order Richardson</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>4945</td>
<td>1220</td>
<td>815.5</td>
<td>667.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>8625</td>
<td>1722.5</td>
<td>925</td>
<td>607.5</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>8085</td>
<td>1666.5</td>
<td>985.5</td>
<td>742.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>15985</td>
<td>2385</td>
<td>1295</td>
<td>810</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>10585</td>
<td>2301</td>
<td>1291.5</td>
<td>817.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>24035</td>
<td>3312.5</td>
<td>1480</td>
<td>810</td>
</tr>
</tbody>
</table>
TABLE 4.41

Mildly non-linear equations, problem II. Work per point

<table>
<thead>
<tr>
<th>h^{-1}</th>
<th>Strategy</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simultaneous displacement</td>
<td>Second-order Richardson</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>16130</td>
<td>3555</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>13500</td>
<td>1997.5</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>29902.5</td>
<td>4323</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>27500</td>
<td>2937.5</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>45342.5</td>
<td>6203</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>45200</td>
<td>4582.5</td>
</tr>
</tbody>
</table>
TABLE 4.42

Mildly non-linear equations, problem II (linearised)

Work per point

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Strategy</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Simultaneous displacement</td>
<td>Second-order Richardson</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>15152.5</td>
<td>3233</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>13095</td>
<td>1946.5</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>28310</td>
<td>4123.5</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>28050</td>
<td>2862.5</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>42949.5</td>
<td>5901</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>43949</td>
<td>4465.5</td>
</tr>
</tbody>
</table>
CHAPTER 5:

SUCCESSIVE OVER-RELAXATION AND PRECONDITIONED
ITERATIVE TECHNIQUES APPLIED TO A CERTAIN FORMULATION
OF LAPLACE'S EQUATION IN THE UNIT SQUARE
5.1 INTRODUCTION

The classic forms of block successive over-relaxation are normally used in their implicit form, illustrated in equations (2.35a) to (2.35c) inclusive. It is possible to combine the first two steps, i.e. to solve

\[ \mathbf{B}_r^{-1} \mathbf{A}_r \mathbf{z}^{(n+1)} + \mathbf{B}_r^{-1} \mathbf{C}_r \mathbf{z}^{(n)} = \mathbf{B}_r^{-1} \mathbf{b}_r \]  

for \( \mathbf{z}^{(n+1)} \). In the solution of elliptic partial differential equations, the sub-matrices \( \mathbf{A}_r, \mathbf{B}_r \) and \( \mathbf{C}_r \) are very sparse, so that the form of the iteration given in Chapter 2 requires very little work per iteration. However, the matrices \( \mathbf{B}_r^{-1} \) (and hence multiples of them) are not sparse, so that the alternative form of the iteration given in equation (5.1) requires an excessive amount of work. As an illustration, consider the solution of the model problem of Chapter 2 (Laplace's equation in a unit square). If the usual five-point finite-difference equation is applied to each of the \( n^2 \) (uniform) mesh points in line ordering, and the resulting equations solved by the line SOR method, then equations (2.35a) and (2.35b) require in general 4 additions and 4 multiplications per point per iteration, whilst the alternative form (5.1) requires 2n additions and 2n multiplications per point per iteration assuming that the values of \( \mathbf{B}_r^{-1} \mathbf{A}_r, \mathbf{B}_r^{-1} \mathbf{C}_r \) and \( \mathbf{B}_r^{-1} \mathbf{b}_r \) are known. If \( n > 2 \), the implicit form is the more efficient of the two, and the larger the value of \( n \), the greater the advantage in using the implicit form.

This chapter presents the novel approach of using a very small block of fixed size. Thus the loss of sparseness when the explicit form of block SOR is used is small and independent of the size of the problem. It is shown that, with suitable programming, the explicit form of block SOR actually solves the model problem of Chapter 2 with much less computational effort than is needed by the implicit form.

This explicit block SOR method can be considered as a point SOR method.
applied to a transformed matrix. It can be shown that this matrix is positive definite and so methods of preconditioned simultaneous iteration can be applied to the solution of the problem. Experimental results are presented for these methods.
5.2 THE FOUR-POINT BLOCK

Consider the model problem in the unit square, with the mesh points ordered in groups of four, the groups themselves being ordered in red-black (chess-board) ordering, as shown in Figure 5.1.

Suppose that the problem under consideration is rather more general than the model problem, so that the left-hand side of the finite difference equation has the form

$$z_{i,j} + a_1 z_{i-1,j} + a_2 z_{i,j+1} + a_3 z_{i+1,j+1} + a_4 z_{i,j-1} = 0.$$  \hspace{1cm} (5.2)

This can be represented diagrammatically as a stencil to be applied to each of the mesh points in turn, as shown in Figure 5.2.
The resulting coefficient matrix, illustrated here for an 8×8 mesh, has the block structure

\[
A = \begin{bmatrix}
R_0 & R_0 & R_0 & R_0 & 0 & R_2 & R_3 & 0 \\
0 & R_0 & R_0 & 0 & R_0 & R_2 & R_3 & R_3 \\
0 & R_0 & R_0 & 0 & R_0 & 0 & R_2 & R_3 \\
R_1 & R_2 & R_3 & R_3 & 0 & R_0 & R_0 & R_0 \\
R_1 & R_2 & R_3 & R_3 & 0 & R_0 & R_0 & R_0 \\
R_1 & R_2 & R_3 & R_3 & 0 & R_0 & R_0 & R_0 \\
0 & R_1 & R_2 & R_3 & 0 & R_0 & R_0 & R_0 \\
0 & R_1 & R_2 & R_3 & 0 & R_0 & R_0 & R_0
\end{bmatrix}
\]
where

\[
R_0 = \begin{bmatrix}
1 & \alpha_2 & 0 & \alpha_3 \\
\alpha_4 & 1 & \alpha_3 & 0 \\
0 & \alpha_1 & 1 & \alpha_4 \\
\alpha_1 & 0 & \alpha_2 & 1 \\
\end{bmatrix}
\]

\[
R_1 = \begin{bmatrix}
0 & 0 & 0 & \alpha_1 \\
0 & 0 & \alpha_1 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
R_2 = \begin{bmatrix}
0 & 0 & 0 & 0 \\
\alpha_2 & 0 & 0 & 0 \\
0 & 0 & 0 & \alpha_2 \\
0 & 0 & 0 & 0 \\
\end{bmatrix}
\]

\[
R_3 = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & \alpha_3 & 0 & 0 \\
\alpha_3 & 0 & 0 & 0 \\
\end{bmatrix}
\]

and

\[
R_4 = \begin{bmatrix}
0 & \alpha_4 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & \alpha_4 & 0 \\
\end{bmatrix}
\]

Unlike the block iterative schemes discussed in Chapter 2, the coefficient matrix here is not block tridiagonal. However, because the blocks are taken in red-black ordering, the matrix has block property (A) and is block consistently ordered. If the blocks are taken in natural ordering, the matrix is also block consistently ordered. It follows immediately that the full theory of block SOR applies for both orderings, and that they yield the same rate of convergence for the same amount of work. Here we have chosen to work with the red-black ordering.
To derive the explicit block SOR method for this problem, it is necessary to calculate the matrix

$$A^E = \left[ \text{diag}(R_0) \right]^{-1} A. \quad (5.4)$$

The matrix $[\text{diag}(R_0)]^{-1}$ is simply $\text{diag}(R_0^{-1})$, and the inverse of $R_0$ is given by,
\[
\begin{bmatrix}
1 - \alpha_1 \alpha_3 - \alpha_2 \alpha_4 & \alpha_2 (\alpha_2 \alpha_4 - \alpha_1 \alpha_3 - 1) & 2 \alpha_2 \alpha_3 & \alpha_3 (\alpha_1 \alpha_3 - \alpha_2 \alpha_4 - 1) \\
\alpha_4 (\alpha_2 \alpha_4 - \alpha_1 \alpha_3 - 1) & 1 - \alpha_1 \alpha_3 - \alpha_2 \alpha_4 & \alpha_3 (\alpha_1 \alpha_3 - \alpha_2 \alpha_4 - 1) & 2 \alpha_3 \alpha_4 \\
2 \alpha_1 \alpha_4 & \alpha_1 (\alpha_1 \alpha_3 - \alpha_2 \alpha_4 - 1) & 1 - \alpha_1 \alpha_3 - \alpha_2 \alpha_4 & \alpha_4 (\alpha_2 \alpha_4 - \alpha_1 \alpha_3 - 1) \\
\alpha_1 (\alpha_1 \alpha_3 - \alpha_2 \alpha_4 - 1) & 2 \alpha_1 \alpha_2 & \alpha_2 (\alpha_2 \alpha_4 - \alpha_1 \alpha_3 - 1) & 1 - \alpha_1 \alpha_3 - \alpha_2 \alpha_4 \\
\end{bmatrix}
\]

\[ R_0^{-1} = \frac{1}{d} \]

\[ (5.5a) \]
where
\[ d = (a_1 a_3 - a_2 a_4)^2 - 2(a_1 a_3 * a_2 a_4) + 1. \] (5.5b)

The block structure of \( A^E \) is the same as that of \( A \) (equation (5.3a)), with the submatrices \( R_0 \) replaced by identity matrices, and the submatrices \( R_i \), \( i=1,2,3,4 \) replaced by \( R^{-1}_0 R_i \). Where \( R_i \) has a column of zeros, so does \( R^{-1}_0 R_i \), and where an element \( a_i \) occurs as the \((p,q)^{th}\) element of \( R_i \), the \( q^{th} \) column of \( R^{-1}_0 R_i \) is the \( p^{th} \) column of \( R^{-1}_0 \), multiplied by \( a_i \). So, for example,

\[
R^{-1}_0 R_1 = \frac{1}{d} \begin{bmatrix}
0 & 0 & a_1 a_2 (a_2 a_4 - a_1 a_3 - 1) & a_1 (1 - a_1 a_3 - a_2 a_4) \\
0 & 0 & a_1 (1 - a_1 a_3 - a_2 a_4) & a_1 a_4 (a_2 a_4 - a_1 a_3 - 1) \\
0 & 0 & a_1 (a_1 a_3 - a_2 a_4 - 1) & 2 a_1 a_4 \\
0 & 0 & 2 a_1 a_2 & a_1 (a_1 a_3 - a_2 a_4 - 1) \\
\end{bmatrix}
\] (5.6)

For the model problem,

\[ a_1 = a_2 = a_3 = a_4 = -\frac{1}{4}, \]

so that

\[
R^{-1}_0 = \frac{1}{6} \begin{bmatrix}
7 & 2 & 1 & 2 \\
2 & 7 & 2 & 1 \\
1 & 2 & 7 & 2 \\
2 & 1 & 2 & 7 \\
\end{bmatrix}
\] (5.7)

and, for example,

\[
R^{-1}_0 R_1 = \frac{1}{24} \begin{bmatrix}
0 & 0 & 2 & 7 \\
0 & 0 & 7 & 2 \\
0 & 0 & 2 & 1 \\
0 & 0 & 1 & 2 \\
\end{bmatrix}
\] (5.8)

We shall now show that the explicit form of this block SOR method requires less work per iteration than the implicit form (for the model problem).
The most efficient way of implementing the implicit form is to iterate with a transformed vector and use a normalised LU decomposition algorithm to solve the blocks of four equations. The coefficient matrix of the block has the same structure as the block matrix for peripheral SOR. In Chapter 2 it is shown that this block can be solved using 4 additions and 4 multiplications per unknown. These figures neglect some of the variations in the algorithm for the first and last elements in the vector of unknowns. This makes little difference when the block size is large, but the variations cannot be neglected when dealing with a block of only 4 unknowns. In fact, such a block can be solved using only 12 additions and 12 multiplications, an average of 3 additions and 3 multiplications per point. Taking into consideration the work needed to calculate the right-hand-side of each block of 4 equations and the over-relaxation process, the implicit form of this block SOR requires an average of 7 additions and 6 multiplications per point per iteration.

The coefficient matrix of the explicit form of the block SOR method has in general 8 non-zero off-diagonal elements per row, so that each iteration requires 10 additions and 9 multiplications per point, including the work associated with the over-relaxation process. However, not all 8 elements are different, and making use of this reduces the work requirements to 10 additions and 4 multiplications per point.

Further savings in work can be made if the equations are dealt with in groups of four. Consider the area of mesh shown in Figure 5.3.

\[
\begin{array}{c|c|c|c}
  z_G & z_F \\
  \hline
  z_H & z_4 & z_3 \\
  \hline
  z_A & z_1 & z_2 \\
  z_B & z_C \\
\end{array}
\]

**FIGURE 5.3**
The equations for the calculation of the values $z_1, z_2, z_3$ and $z_4$ contain certain expressions involving $z_A$ to $z_H$ inclusive in common, and these need only be calculated once. Thus if we set

$$s_1 = z_A + z_B,$$
$$s_2 = z_C + z_D,$$
$$s_3 = z_E + z_F,$$
$$s_4 = z_G + z_H,$$
$$s_5 = s_1 + s_3 + s_3 + s_3,$$
$$s_6 = s_2 + s_2 + s_4 + s_4,$$

and then we have

$$z_1 = \frac{1}{24} (7s_1 + s_6 + s_3),$$
$$z_2 = \frac{1}{24} (7s_2 + s_5 + s_4),$$
$$z_3 = \frac{1}{24} (7s_3 + s_6 + s_1),$$
$$z_4 = \frac{1}{24} (7s_4 + s_5 + s_2).$$

With the addition of the over-relaxation process, these equations require an average of $6\frac{1}{2}$ additions and 3 multiplications per point per iteration (assuming that the constant $\frac{1}{24}$ is stored). This is both less additions and less multiplications than the implicit form of the method.

Even further savings in work can be made by solving equations (5.9b) and (5.9d), and using the values of $z_1$ and $z_3$ thus found to determine $z_2$ and $z_4$. We calculate the values of $s_1, s_2, s_3, s_4$ and $s_6$, then find $z_1$ and $z_3$ as before, and set

$$s_7 = z_1 + z_3.$$

Now we have

$$z_2 = \frac{1}{24} \left(7s_2 + s_5 + s_4\right)$$
$$= \frac{1}{12} s_1 + \frac{7}{24} s_2 + \frac{1}{12} s_3 + \frac{1}{24} s_4$$
$$= \frac{1}{4} \left(s_2 + \left\{\frac{7}{24} s_1 + \frac{1}{12} s_2 + \frac{1}{24} s_3 + \frac{1}{12} s_4\right\}\right)$$
\[ \begin{aligned}
&= \left( \frac{7}{24} s_3 + \frac{1}{12} s_2 + \frac{1}{24} s_1 + \frac{1}{12} s_4 \right) \\
&= \frac{1}{4} (s_2 + \frac{1}{24} (7s_1 + s_6 + s_3) + \frac{1}{24} (7s_3 + s_6 + s_1)) \\
&= \frac{1}{4} (s_2 + z_1 + z_3) \\
&= \frac{1}{4} (s_2 + s_7) \\
\end{aligned} \]

and similarly it can be shown that

\[ z_4 = \frac{1}{4} (s_4 + s_7) \]  

The average work per point for this implementation of the method, including the over-relaxation process, is only \( \frac{51}{2} \) additions and \( \frac{21}{2} \) multiplications. On the assumption that a multiplication takes 2.5 times as much computer time as an addition, the explicit form of this block SOR method takes only about half as much work as the implicit method.
5.3 EXPERIMENTAL RESULTS FOR THE FOUR-POINT BLOCK SOR METHOD

The experimental results presented here are for the equation $\nabla^2 \phi = 0$ in the unit square with zero boundary conditions. The initial estimate $z^{(0)}$ of the solution has all elements unity in every case.

An attempt was made to determine the eigenvalues of the Jacobi iteration matrix, $B$, algebraically, by factorising the determinant of $(B - \lambda I)$ for the case $h^{-1} = 5$. The expression could not be completely factorised, and although expressions for some of the eigenvalues were found, these did not include the eigenvalue of largest magnitude. The spectral radii $\rho(B)$ given here are obtained experimentally using the power method, and the optimum values of $\omega$ computed from them using the usual formula (equation (2.28)).

Two estimates are made of the number of iterations required. One assumes that only one eigenvalue has the magnitude of the spectral radius (equation (2.20')); the other assumes that there are two eigenvalues of this magnitude (equation (2.21)). These are labelled Theory 1 and Theory 2 respectively in the following table. The equations are deemed to have converged when the maximum absolute difference between corresponding elements of successive iterates is less than $5 \times 10^{-6}$.

(For this method, there must be an even number of mesh points in both directions. Hence numerical results are given for odd values of $h^{-1}$).

The most noteworthy point to arise from Table 5.1 is that the values of $\rho(B)$ are very close to the values of $\rho(B)$ for the line Jacobi matrix (Chapter 2). The predicted number of iterations is the same for both methods for $h^{-1} = 5, 15, 25$ and 35, and the corresponding theoretical optimum values of $\omega$ agree to 2 decimal places. Thus, although no theoretical value is known for the spectral radius of the four-point block Jacobi matrix, the value for the corresponding line Jacobi matrix, i.e.

$$\rho(B) = \frac{\cosh h}{2 - \cosh h},$$  \hspace{1cm} (5.11)

provides a good enough estimate for practical purposes.
### TABLE 5.1

Laplace's equation in the unit square. Four-point block SOR

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\rho(B)$</th>
<th>Iterations</th>
<th>Optimum $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Theory 1</td>
<td>Theory 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Experiment 1</td>
<td>Experiment 2</td>
</tr>
<tr>
<td>5</td>
<td>0.666667</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>15</td>
<td>0.957126</td>
<td>20</td>
<td>28</td>
</tr>
<tr>
<td>25</td>
<td>0.984344</td>
<td>34</td>
<td>47</td>
</tr>
<tr>
<td>35</td>
<td>0.991976</td>
<td>48</td>
<td>66</td>
</tr>
</tbody>
</table>
In practice, the four-point block SOR method converges faster than the line SOR method. The behaviour of the four-point block method is closely described by Theory 1 whilst the behaviour of the line method is very close to Theory 2.

The computational effort per point per iteration required by four-point block SOR is (on average) $\frac{1}{2}$ additions, $\frac{1}{2}$ multiplications and 1 convergence test. The weighting scheme developed in Chapter 2 gives this a relative execution time of 13.25. SLOR, which requires the least effort per point per iteration of the usual block methods, has a relative execution time of 20.0. It follows from this and the previous paragraph that this new block SOR method is more efficient than SLOR.

Table 5.2 gives the relative execution times per point of the new method. The execution times per point of two-line SOR, the most efficient of the four forms of SOR discussed in Chapter 2, are presented for comparison.

**TABLE 5.2**

Execution times per point for two-line SOR and four-point block SOR

<table>
<thead>
<tr>
<th>Method</th>
<th>S2LOR</th>
<th>Four-point block SOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h = 1$</td>
<td>164.5</td>
<td>119.25</td>
</tr>
<tr>
<td>5</td>
<td>423</td>
<td>333.75</td>
</tr>
<tr>
<td>15</td>
<td>705</td>
<td>516.75</td>
</tr>
<tr>
<td>25</td>
<td>1010.5</td>
<td>694</td>
</tr>
</tbody>
</table>

This table indicates that four-point block SOR requires less computer time than the usual block SOR methods to solve Laplace's equation in the unit square. It also requires less storage; whereas the usual methods use large
arrays in solving the blocks of equations, the new methods needs negligible extra storage beyond the single vector of unknowns. A corollary of this is that the new method is very much simpler to program than the classic block SOR methods.
5.4 EXPERIMENTAL RESULTS FOR PRECONDITIONED METHODS OF SIMULTANEOUS ITERATION APPLIED TO THE EXPLICIT FOUR-POINT BLOCK MATRIX

The explicit four-point block matrix, $A^E$, defined by equation (5.4) can be used in preconditioned methods of simultaneous iteration. Since for the model problem, $A^E$ is symmetric, irreducibly diagonally dominant and has positive diagonal elements, it is positive definite and the full theory of preconditioning as given in Chapter 2 applies.

The matrix $A^E$ has property (A) and $\sigma_1$-ordering, which implies that the optimum preconditioning parameter is unity (Theorem 2.2). Evans (1968), p.303 further shows that the $P$-condition number of the matrix preconditioned with this parameter is

$$P(1) = \frac{1}{1 - [\rho(B)]^2}, \quad (5.12)$$

where $\rho(B)$ is the spectral radius of the Jacobi iteration matrix derived from $A^E$.

Consider the matrix, $C$ say, which is formed by taking the blocks of four equations in natural (line-by-line) ordering. The corresponding explicit block matrix $C^E$ is then

$$C^E = [\text{diag}(R_0)]^{-1} C. \quad (5.13)$$

$C^E$ is positive definite and has property (A), but has $\sigma_2$-ordering. Therefore its $P$-condition number can be minimised by preconditioning with a parameter $1 < \omega < 2$. We note that the matrices $A^E$ and $C^E$ have the same eigenvalues but different eigenvectors.

Table 5.3 shows the effect of preconditioning with the optimum preconditioning parameter on the matrices $A^E$ and $C^E$. The notation used is that introduced in Chapter 2.

For this problem and range of values of $h^{-1}$, the minimum $P$-condition number which can be achieved by preconditioning $A^E$ is about twice the minimum which can be achieved by preconditioning $C^E$. For the remainder of this section we shall consider only the latter.
**TABLE 5.3**
Explicit four-point block. Eigenvalues of coefficient matrices with and without preconditioning

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Matrices $A^E$ and $C^E$ with and without preconditioning</th>
<th>Matrix $C^E$ with preconditioning with optimum parameter</th>
<th>Matrix $A^E$ with preconditioning with optimum parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\lambda_1$</td>
<td>$\lambda_N$</td>
<td>$P(0)$</td>
</tr>
<tr>
<td>5</td>
<td>1.666667</td>
<td>0.333333</td>
<td>5.00</td>
</tr>
<tr>
<td>15</td>
<td>1.957126</td>
<td>0.042874</td>
<td>45.65</td>
</tr>
<tr>
<td>25</td>
<td>1.984301</td>
<td>0.015659</td>
<td>126.72</td>
</tr>
<tr>
<td>35</td>
<td>1.991853</td>
<td>0.008018</td>
<td>248.43</td>
</tr>
</tbody>
</table>
Preconditioning the matrix \( C^E \) with the optimum parameter reduces its P-condition number such that

\[ P(\omega_0) = O(\sqrt{P(0)}) \quad (5.14) \]

as expected. The exact value is about \( 0.7\sqrt{P(0)} \) for \( h^{-1} = 5 \), rising to about \( 2\sqrt{P(0)} \) for \( h^{-1} = 35 \). It is reasonable to assume that the multiple of \( \sqrt{P(0)} \) increases steadily as \( h^{-1} \) increases. This is a disappointing result; in other experiments recorded in this thesis (Chapters 2 and 4) the P-condition number is reduced to approximately 0.5 or 0.6 times \( \sqrt{P(0)} \), and the multiple does not increase significantly as \( h^{-1} \) increases.

The following tables record the predicted and actual behaviour of the simultaneous displacement and second-order Richardson methods with and without preconditioning. The parameters \( \alpha \) and \( \beta \) are calculated from the maximum and minimum eigenvalues as described in Chapter 2. The initial estimate \( z^{(0)} \) of the solution vector has all elements unity in every case. The equations are deemed to have converged when the maximum absolute difference between corresponding elements of successive iterates is less than \( 5 \times 10^{-6} \).

Preconditioning clearly makes a substantial reduction in the number of iterations required by these two methods of simultaneous iteration. The improvements, however, are not as marked as in other preconditioning experiments described in this thesis, because the reduction in the P-condition number is somewhat less here.

In Table 5.4, the predicted number of iterations for simultaneous displacement without preconditioning exceeds the actual number in every case. This phenomenon has also been recorded in connection with Mikhlin and Smolitsky's finite-difference equation in Chapter 2, where a possible explanation is given.

The Chebyshev acceleration and Chebyshev semi-iterative methods, with and without preconditioning, can clearly be applied to this problem. However, the results do not differ significantly from those for the second-order Richardson method, Table 5.5.
**TABLE 5.4**

Explicit four-point block. Experimental results for the simultaneous displacement method

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Without preconditioning</th>
<th></th>
<th>With preconditioning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>Iterations</td>
<td>$\alpha$</td>
<td>Iterations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Theory</td>
<td>Experiment</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.0000</td>
<td>30</td>
<td>29</td>
<td>1.1994</td>
</tr>
<tr>
<td>15</td>
<td>1.0000</td>
<td>279</td>
<td>218</td>
<td>1.6613</td>
</tr>
<tr>
<td>25</td>
<td>1.0000</td>
<td>773</td>
<td>541</td>
<td>1.7187</td>
</tr>
<tr>
<td>35</td>
<td>1.0000</td>
<td>1516</td>
<td>978</td>
<td>1.6302</td>
</tr>
</tbody>
</table>
TABLE 5.5

Explicit four-point block. Experimental results for the second-order Richardson method

<table>
<thead>
<tr>
<th>h^{-1}</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>α</td>
<td>β</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1.1459</td>
<td>0.1459</td>
</tr>
<tr>
<td>15</td>
<td>1.5508</td>
<td>0.5508</td>
</tr>
<tr>
<td>25</td>
<td>1.7003</td>
<td>0.7003</td>
</tr>
<tr>
<td>35</td>
<td>1.7757</td>
<td>0.7757</td>
</tr>
</tbody>
</table>
A case occurred during experiments with this problem where the choice of the initial estimate of the solution directly affected the behaviour of the iteration. The model problem with $h^{-1}=5$ was being solved by simultaneous displacement (without preconditioning) for a range of values of $\alpha$, in order to verify the prediction from theory that the iteration converges most rapidly when $\alpha=1$ and is divergent for $\alpha>1.5$. Instead, the number of iterations decreased as $\alpha$ increased from 0.8, reached a minimum at $\alpha=1.5$ and then increased very rapidly. Estimates of the spectral radii of the iteration matrix for $1\leq\alpha\leq1.5$, made from the number of iterations required, indicated that the iteration was being governed in this range by the largest subdominant eigenvalue, as if the estimate of the solution was deficient in the dominant eigenvector. When an initial estimate of the solution composed of random numbers was used instead of all ones, the iteration behaved according to theory.

Table 5.6 shows the computational effort per point for the converged problem for the simultaneous displacement and second-order Richardson method. This table assumes that the problem is more general than the model problem, and that all the non-zero off-diagonal elements in a row of the coefficient matrix are different. The figures for the work per point per iteration are taken from Table 2.14 of Chapter 2.

Table 5.6 shows the poor saving in effort achieved by the use of preconditioning in this problem. For the second-order Richardson method, for example, the work is only reduced by a factor of about 20%. In the application of the second-order Richardson method to other problems in this thesis, savings of 50% to 70% have been made by the use of preconditioning.
TABLE 5.6

Explicit four-point block. Work per point for the simultaneous displacement and second-order Richardson methods

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simultaneous displacement</td>
<td>Second-order Richardson</td>
</tr>
<tr>
<td>5</td>
<td>957</td>
<td>584</td>
</tr>
<tr>
<td>15</td>
<td>7194</td>
<td>1715.5</td>
</tr>
<tr>
<td>25</td>
<td>17853</td>
<td>2774</td>
</tr>
<tr>
<td>35</td>
<td>32274</td>
<td>3796</td>
</tr>
</tbody>
</table>
If the problem to be solved is the model problem, not all the non-zero off-diagonal elements in a row of the coefficient matrix are different and a substantial amount of computational effort can be saved by utilising this fact. The numbers of operations can be reduced from those given in Chapter 2 as follows:

Simultaneous displacement: 9 additions
4 multiplications
1 convergence test
(Weighted total 20.5)

Simultaneous displacement with preconditioning: 17 additions
8 multiplications
1 convergence test
(Weighted total 39.5)

Second-order Richardson: 10 additions
5 multiplications
1 convergence test
(Weighted total 24)

Second-order Richardson with preconditioning: 18 additions
9 multiplications
1 convergence test
(Weighted total 42).

The estimates of work per point for the converged problem using these figures are given in Table 5.7.

Although these figures for computational effort are considerably less than those of Table 5.6, they are very much larger than the figures for four-point explicit block SOR. Therefore this form of block preconditioning method is not recommended for solving this particular problem.
**TABLE 5.7**  
Explicit four-point block. Work per point for the model problem using the simultaneous displacement and second-order Richardson methods.

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Simultaneous displacement</td>
<td>Second-order Richardson</td>
</tr>
<tr>
<td>5</td>
<td>594.5</td>
<td>384</td>
</tr>
<tr>
<td>15</td>
<td>4469</td>
<td>1128</td>
</tr>
<tr>
<td>25</td>
<td>11090.5</td>
<td>1824</td>
</tr>
<tr>
<td>35</td>
<td>20049</td>
<td>2496</td>
</tr>
</tbody>
</table>
CHAPTER 6

AN APPROXIMATE METHOD OF PRECONDITIONED SIMULTANEOUS DISPLACEMENT
6.1 INTRODUCTION

The introduction to Chapter 5 illustrates why the explicit forms of block matrices are not normally used in block iterative schemes, namely that these forms are not nearly as sparse as the matrices used in the implicit iteration. Sparse approximations to these explicit forms, can, however, be used as the basis of iterative methods, provided the methods include some correction process which forces them to converge to the correct solution. This chapter explores the application of preconditioned simultaneous displacement to such a method based on the explicit line matrix, and presents some experimental results.
6.2 THE EXPLICIT LINE BLOCK MATRIX

Let us consider the model problem of Chapter 2, with the $n^2$ mesh points in natural (line-by-line) ordering as shown in Figure 2.2. The coefficient matrix $A$ of the resulting finite-difference equations, normalised so that its diagonal elements are unity, has the block tridiagonal form:

\[
A = \begin{bmatrix}
B_1 & C_1 & 0 & \cdots & 0 \\
A_2 & B_2 & C_2 & \cdots & 0 \\
0 & A_3 & B_3 & C_3 & \cdots \\
& \ddots & \ddots & \ddots & \ddots \\
& & 0 & A_{n-1} & B_{n-1} & C_{n-1} \\
& & & 0 & A_n & B_n
\end{bmatrix},
\]  

(6.1a)

where

\[
B_i = \begin{bmatrix}
1 & -0.25 & & & & & \\
-0.25 & 1 & -0.25 & & & & \\
& -0.25 & 1 & -0.25 & & & \\
& & \ddots & \ddots & \ddots & & \\
& & & \ddots & \ddots & \ddots & \\
& & & & \ddots & \ddots & \ddots \\
& & & & & \ddots & \ddots \\
& & & & & & 1
\end{bmatrix},
\]

(6.1b)

and $C_i = A_{i+1} = \begin{bmatrix}
-0.25 & 0 \\
0 & -0.25
\end{bmatrix}$, $i = 1, 2, \ldots, n-1,$  

(6.1c)

the sub-matrices $A_i, B_i$ and $C_i$ being of order $n$.

To derive the explicit block form of this matrix ($A^E$ say), it is necessary to calculate
\[ A^E = \text{diag}\{B_1, B_2, \ldots, B_n\}^{-1}A \]

\[
= \text{diag}\{B_1^{-1}, B_2^{-1}, \ldots, B_n^{-1}\}A. \quad (6.2)
\]

The matrix \( A^E \) has the same block structure as \( A \), with the sub-matrices \( B_i \) replaced by identity matrices, and the sub-matrices \( A_i \) and \( C_i \) replaced by \( B_i^{-1}A_i \) and \( B_i^{-1}C_i \) respectively.

The value of \( D=B_i^{-1}, i=1,2,\ldots,n \) is defined as follows. Let \( f(m) \) be the value of the determinant of a matrix of the form of \( B_i \) and of order \( m \). We have

\[
f(1) = 1
\]

and if we define

\[
f(0) = 1,
\]

it can readily be proved by induction that

\[
f(m) = f(m-1) - 0.25 f(m-2), \quad m \geq 2. \quad (6.3)
\]

(It can further be shown that \( f(m) \) is defined explicitly by)

\[
f(m) = 1 + \sum_{j=1}^{\lfloor \frac{m}{2} \rfloor} \frac{(-1)^j (m-j)!}{(m-2j)! j!} (0.25)^j. \quad (6.4)
\]

Then if \( D \) is of order \( n \), we have

\[
d_{j,k} = \frac{(0.25)^{\lfloor j-k \rfloor} f(x)f(y)}{f(n)}, \quad j=1,2,\ldots,n, \quad k=1,2,\ldots,n, \quad (6.5a)
\]

where

\[
x = \begin{cases} \max(j-1,n-k) & j<k \\ \max(n-j,k-1) & j \geq k \end{cases}
\]

and

\[
y = \min[\min(j,n-j+1),\min(k,n-k+1)]-1. \quad (6.5b)
\]

The elements of the matrices \( B_i^{-1}A_i \) and \( B_i^{-1}C_i \) are \(-0.25\) times the corresponding elements of \( D \).

Equation (6.4) is derived from results given by Rutherford (1968), pp 229-236, and Rutherford (1951), pp 232-241. Equation (6.5) is equivalent to a definition given by Kershaw (1969), pp 189-191, in which the elements \( d_{j,k} \) are given in terms of Chebyshev polynomials of the second kind.
6.3 PRECONDITIONING AND APPROXIMATE PRECONDITIONING

The matrix $A^E$ defined in the previous section is symmetric with positive diagonal elements, and can be shown to be irreducibly diagonally dominant. It is therefore positive definite and the full theory of preconditioning applies. Table 6.1 shows the reduction in the $P$-condition number when this matrix is preconditioned with the optimum parameter $\omega^*_b$, found experimentally to within ±0.05. The notation used is that of Chapter 2.

The reductions made in the $P$-condition number are very substantial indeed in this experiment. We have:

- for $h^{-1} = 5$, $P(\omega^*_b) = 0.61 \sqrt{P(0)}$,
- for $h^{-1} = 10$, $P(\omega^*_b) = 0.47 \sqrt{P(0)}$

and for $h^{-1} = 20$, $P(\omega^*_b) = 0.41 \sqrt{P(0)}$,

so that, for this range of values of $h^{-1}$, the multiple of $\sqrt{P(0)}$ actually decreases as $h^{-1}$ increases. No attempt was made to obtain experimental results for values of $h^{-1} > 20$ because of the prohibitively large amount of computer time which would have been required. Unfortunately because of the lack of sparseness of the coefficient matrix, methods of preconditioned simultaneous iteration would be grossly inefficient, even with these large reductions in the $P$-condition number.

A possible approach is to replace the matrices $L$ and $U$ in the preconditioning process

$$B_\omega = (I-\omega U)^{-1}(I-\omega L)^{-1}A^E$$

(6.6)

(where $A^E = I - L - U$) by sparse matrices $\tilde{L}$ and $\tilde{U}$ which in some sense approximate the original matrices. Now $L$ and $U$ have the block structures

$$L = \begin{bmatrix} 0 & 0 & 0 \\ E & 0 & 0 \\ 0 & E & 0 \end{bmatrix}$$

(6.7a)
**TABLE 6.1**

Explicit line block Eigenvalues of coefficient matrices with and without preconditioning

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\lambda_1$</td>
<td>$\lambda_n$</td>
</tr>
<tr>
<td>5</td>
<td>1.679285</td>
<td>0.320715</td>
</tr>
<tr>
<td>10</td>
<td>1.906681</td>
<td>0.093319</td>
</tr>
<tr>
<td>20</td>
<td>1.975677</td>
<td>0.024323</td>
</tr>
</tbody>
</table>
and $U = \begin{bmatrix}
0 & E & 0 \\
0 & E & 0 \\
0 & 0 & E \\
0 & 0 & 0 \\
\end{bmatrix}$, \quad (6.7b)

where

\[
E = -B_1^{-1}A_1 = -B_2^{-1}A_2 = \ldots = -B_n^{-1}A_n \\
= -B_1^{-1}C_1 = -B_2^{-1}C_2 = \ldots = -B_n^{-1}C_n.
\] \quad (6.7c)

The elements $e_{j,k}$ of $E$, where

\[
e_{j,k} = -0.25d_{j,k}, \quad j=1,2,\ldots,n, \quad k=1,2,\ldots,n,
\]

(\(d_{j,k}\) is defined in equation (6.5)), have a number of interesting properties. Numerical studies indicate that an element $e_{j,k}$ of an $(n \times n)$ matrix approaches a limit as $n \to \infty$. Furthermore, the element of maximum modulus in any diagonal (including super- and sub-diagonal) row occurs at the centre point of the row (or centre two points if there are an even number of points in the row). The value of this element also approaches a limit as the matrix size tends to infinity. As expected from the presence of the $0.25|j-k|^{-1}$ term in $e_{j,k}$, the magnitude of $e_{j,k}$ is inversely proportional to $|j-k|$. In other words, the further an element is from the leading diagonal, the smaller is its magnitude.

These properties allow us to deduce approximate upper bounds for the element of largest modulus in any given diagonal. The upper bound for the leading diagonal is about $0.3$, for the first super- and sub-diagonals about $0.08$ and for the second super- and sub-diagonals, about $0.022$. Upper bounds for subsequent diagonals decrease monotonically.

If we let,
then $\tilde{E}$ is an approximation to $E$ in the sense that the zero elements of $\tilde{E}$ correspond to elements of $E$ not greater than 0.022.

Therefore, if the matrices

$$
\tilde{L} = 
\begin{bmatrix}
0 & 0 & 0 \\
\tilde{E} & 0 & 0 \\
\tilde{E} & \tilde{E} & 0 \\
0 & 0 & 0 \\
\tilde{E} & \tilde{E} & \tilde{E} \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix}
$$

(6.10a)

and

$$
\tilde{U} = 
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
\tilde{E} & \tilde{E} & 0 \\
0 & 0 & 0 \\
\tilde{E} & \tilde{E} & \tilde{E} \\
0 & 0 & 0 \\
0 & 0 & 0
\end{bmatrix}
$$

(6.10b)

replace the matrices $L$ and $U$ in the preconditioning equation (6.6), the behaviour of the preconditioning should not change very much, but a considerable amount of work will be saved.

Table 6.2 gives experimental results for this approximate preconditioning.
**TABLE 6.2**

Explicit line block. Eigenvalues of coefficient matrices with approximate preconditioning

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>$\omega_b$</th>
<th>$\lambda_1(\omega_b)$</th>
<th>$\lambda_m(\omega_b)$</th>
<th>$P(\omega_b)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1.20</td>
<td>1.038956</td>
<td>0.745529</td>
<td>1.39</td>
</tr>
<tr>
<td>10</td>
<td>1.60</td>
<td>1.663829</td>
<td>0.584682</td>
<td>2.85</td>
</tr>
<tr>
<td>20</td>
<td>1.90</td>
<td>3.257981</td>
<td>0.522209</td>
<td>6.24</td>
</tr>
</tbody>
</table>
The table indicates that approximate and exact preconditioning give similar results for small values of $h^{-1}$, but that the similarity decreases quite rapidly as $h^{-1}$ increases. We have:

- for $h^{-1} = 5$, $P(\omega_b) = 0.61 \sqrt{P(0)}$,
- for $h^{-1} = 10$, $P(\omega_b) = 0.63 \sqrt{P(0)}$,
- and for $h^{-1} = 20$, $P(\omega_b) = 0.69 \sqrt{P(0)}$.

Thus approximate preconditioning does not improve the rate of convergence as much as exact preconditioning, but requires only about half as much work per iteration. However, the presence of the explicit line matrix in its exact, non-sparse form in the preconditioning equation means that methods of simultaneous iteration would be inefficient even with the use of approximate preconditioning. To develop an acceptably efficient iterative method based on the explicit line matrix, it is necessary to use a sparse approximation to this matrix.
6.4 A PRECONDITIONED APPROXIMATE METHOD

Consider the matrix $\tilde{A}$ defined by

$$\tilde{A} = I - \tilde{L} - \tilde{U}, \tag{6.11}$$

where $\tilde{L}$ and $\tilde{U}$ are as previously defined. It is symmetric and irreducibly diagonally dominant, therefore it is positive definite and the preconditioning theory applies. Table 6.3 gives results for preconditioning this matrix.

What is interesting here is the very small increase in $P(\omega_b)$ between $h^{-1} = 10$ and $h^{-1} = 20$, as if the curve of $P(\omega_b)$ against $h^{-1}$ flattens out. Unfortunately it was not possible to confirm this, as, despite the sparseness of the matrix, the eigenvalue calculations required an unacceptable large amount of computer time.

The matrix $\tilde{A}$ can be used as the basis of an iterative method to solve the matrix equation

$$A\mathbf{z} = \mathbf{b} \tag{6.12}$$

if some correction term is added to replace the elements dropped from the matrix $A^E$ in forming $\tilde{A}$. Let $b^E$ be the right-hand side of equation (6.12) in its explicit block form, i.e.

$$A^E\mathbf{z} = b^E. \tag{6.13}$$

If we define

$$\tilde{b} = (I - \omega \tilde{U})^{-1} (I - \omega \tilde{L}) b^E, \tag{6.14}$$

then we can define an approximate method of preconditioned simultaneous displacement by

$$z^{(n+1)} = z^{(n)} + \alpha \left( \tilde{b} - (I - \omega \tilde{U})^{-1} (I - \omega \tilde{L})^{-1} \mathbf{z}^{(n)} \right) + [b - A \mathbf{z}^{(n)}], \tag{6.15}$$

for $n=1,2,...$ where $z^{(0)}$ is arbitrary and $b - A \mathbf{z}^{(n)}$ is the correction term. The method should only converge when both expressions in the square parentheses are sufficiently close to zero, which we see from the correction term is when $z^{(n)}$ is close to the solution $\mathbf{z}$ of equation (6.12).
<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Without precondtioning</th>
<th>With precondtioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\lambda_1$</td>
<td>$\lambda_N$</td>
</tr>
<tr>
<td>5</td>
<td>1.650525</td>
<td>0.349475</td>
</tr>
<tr>
<td>10</td>
<td>1.825823</td>
<td>0.174177</td>
</tr>
<tr>
<td>20</td>
<td>1.871312</td>
<td>0.128321</td>
</tr>
</tbody>
</table>

**TABLE 6.3**

Explicit line block. Eigenvalues of approximate coefficient with and without preconditioning
To maximise the rate of convergence of this new method, it is necessary to minimise the spectral radius of the iteration matrix, i.e., to minimise
\[
\rho \left[ I - (I - \omega U)^{-1} (I - \omega L)^{-1} \tilde{A} - A \right].
\]
(6.16)

We require, therefore, to minimise the P-condition number of
\[
(I - \omega U)^{-1} (I - \omega L)^{-1} \tilde{A} + A
\]
(6.17)
by suitable choice of \( \omega \). Clearly this will not be exactly the same as minimising the P-condition number of \((I - \omega U)^{-1} (I - \omega L)^{-1} \tilde{A}\) as illustrated in Table 6.3, although we may expect some similarities in the pattern of results. Table 6.4 shows the P-condition number of expression (6.17) when \( \omega = 0 \) and when \( \omega \) takes its optimum value (determined experimentally to within \( \pm 0.05 \)).

Experiments on the iterative method itself show that neither Table 6.3 nor Table 6.4 provides a particularly accurate guide to the behaviour of the iterative process. The process was applied to the model problem using in one instance zero boundary conditions (Table 6.5) and in another boundary conditions of all 1's (Table (6.6)). The process was carried out using a range of values of \( \omega \) and \( \alpha \), to determine what values of these parameters minimise the number of iterations required. The search for this minimum was made using \( \omega \) in steps of 0.1 and \( \alpha \) in steps of 0.01.
TABLE 6.4

Explicit line block. Eigenvalues of expression (6.17)

<table>
<thead>
<tr>
<th>h^{-1}</th>
<th>Without preconditioning</th>
<th>With preconditioning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \lambda_1 )</td>
<td>( \lambda_N )</td>
</tr>
<tr>
<td>5</td>
<td>3.071813</td>
<td>0.541177</td>
</tr>
<tr>
<td>10</td>
<td>3.220261</td>
<td>0.223523</td>
</tr>
<tr>
<td>20</td>
<td>3.225851</td>
<td>0.140707</td>
</tr>
</tbody>
</table>
**TABLE 6.5**

Experimental results for approximate preconditioned simultaneous displacement with boundary conditions zero

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Minimum number of iterations</th>
<th>Range of values of $\omega$ and $\alpha$ for which number of iterations is minimised</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>9</td>
<td>$\omega$ 1.3 0.54-0.56 1.4 0.53-0.56 1.5 0.52-0.57 1.6 0.52-0.56 1.7 0.53-0.55</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>$\omega$ 1.4 0.59-0.61 1.5 0.55-0.60 1.6 0.54-0.58</td>
</tr>
<tr>
<td>20</td>
<td>11</td>
<td>$\omega$ 1.5 0.57-0.60 1.6 0.56-0.57</td>
</tr>
<tr>
<td>30</td>
<td>11</td>
<td>$\omega$ 1.5 0.57-0.59 1.6 0.56-0.57</td>
</tr>
<tr>
<td>40</td>
<td>11</td>
<td>$\omega$ 1.5 0.57-0.59 1.6 0.56-0.57</td>
</tr>
</tbody>
</table>
TABLE 6.6
Experimental results for approximate preconditioned simultaneous displacement with boundary conditions unity

<table>
<thead>
<tr>
<th>h^{-1}</th>
<th>Minimum number of iterations</th>
<th>Range of values of $\omega$ and $\alpha$ for which number of iterations is minimised</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10</td>
<td>$\omega$ = 1.2, $\alpha$ = 0.56-0.57, $\omega$ = 1.3, $\alpha$ = 0.54-0.57, $\omega$ = 1.4, $\alpha$ = 0.52-0.57, $\omega$ = 1.5, $\alpha$ = 0.52-0.57, $\omega$ = 1.6, $\alpha$ = 0.52-0.57, $\omega$ = 1.7, $\alpha$ = 0.52-0.55</td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>$\omega$ = 1.4, $\alpha$ = 0.58-0.61, $\omega$ = 1.5, $\alpha$ = 0.56-0.60, $\omega$ = 1.6, $\alpha$ = 0.55-0.58</td>
</tr>
<tr>
<td>20</td>
<td>12</td>
<td>$\omega$ = 1.4, $\alpha$ = 0.60-0.61, $\omega$ = 1.5, $\alpha$ = 0.57-0.60, $\omega$ = 1.6, $\alpha$ = 0.57</td>
</tr>
<tr>
<td>30</td>
<td>12</td>
<td>$\omega$ = 1.4, $\alpha$ = 0.60-0.61, $\omega$ = 1.5, $\alpha$ = 0.57-0.60, $\omega$ = 1.6, $\alpha$ = 0.57</td>
</tr>
<tr>
<td>40</td>
<td>12</td>
<td>$\omega$ = 1.4, $\alpha$ = 0.60-0.61, $\omega$ = 1.5, $\alpha$ = 0.57-0.60, $\omega$ = 1.6, $\alpha$ = 0.57</td>
</tr>
</tbody>
</table>
A vital question is whether the method actually converges to the solution of equation (6.12). Since the model problem with zero boundary conditions has a zero solution, the norm of the solution vector produced by the iterative method provides a useful estimate of the accuracy of the solution. In the experiments recorded here the Euclidean norm of the vector is no larger than $O(10^{-5})$. Similar experiments conducted with various SOR methods for the same problem indicates that the new method is at least as accurate as they are.

The work required per point per iteration is:

19 additions
19 multiplications
1 convergence test,

which is much larger than for any SOR method, amounting to an execution time of 68 under our weighting scheme. However, the small number of iterations required by the new method more than compensates for this for large values of $h^{-1}$, as shown in Table 6.7. It appears that the new method is more efficient than even two-line SOR for $h^{-1} \geq 30$, although it requires more storage.

**TABLE 6.7**

Laplace's equation in the unit square. Work per point.

<table>
<thead>
<tr>
<th>$h^{-1}$</th>
<th>Two-line SOR</th>
<th>Approximate preconditioned simultaneous displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>164.5</td>
<td>612</td>
</tr>
<tr>
<td>10</td>
<td>305.5</td>
<td>748</td>
</tr>
<tr>
<td>20</td>
<td>587.5</td>
<td>748</td>
</tr>
<tr>
<td>30</td>
<td>893</td>
<td>748</td>
</tr>
<tr>
<td>40</td>
<td>1151.5</td>
<td>748</td>
</tr>
</tbody>
</table>
It is quite clear from experiments that the new method cannot be fully described by the basic theory of preconditioned simultaneous displacement. The key features of the new method are that the rate of convergence appears to be independent of $h^{-1}$, and that a wide range of values of $\omega$ and $\alpha$ give a minimum number of iterations. Neither of these features has been observed in any other experiment on simultaneous displacement carried out in this thesis.

Conclusion

Experimental results for this method of approximate preconditioned simultaneous displacement indicate that it is both efficient and accurate. However, before recommending it for widespread use further work needs to be done to establish a sound theoretical basis for the strategy.
SUMMARY

This thesis has been concerned with the solution by successive over-relaxation and preconditioned iterative methods of linear and mildly non-linear elliptic partial differential equations. A detailed analysis of the computational efficiency of new and existing algorithms has been made.

It has been demonstrated in Chapter 3 that classic methods of point and block SOR can be applied to problems in two particular non-Cartesian geometries, and that the relative efficiency of the methods is similar to that in Cartesian geometry. It seems likely that the methods could be extended to deal with other non-Cartesian geometries without much difficulty.

A curious anomaly has been pointed out about the discrepancies between the predicted and actual numbers of iterations required by the various classes of SOR. The discrepancies are observed to be a function of the particular SOR method. An investigation into the reasons for this should yield useful information on predicting the numbers of iterations required for iterative methods to converge.

The investigation of mildly non-linear p.d.e.'s in Chapter 4 has shown the applicability of SOR methods to these problems. It has been shown that the choice of inner-outer strategy can substantially affect the amount of work needed to solve the problems. Some factors affecting the best choice of inner/outer iteration strategy have been identified. Numerical studies of other mildly non-linear partial differential equations should lead to the identification of other factors.

The study in Chapter 5 of a four-point block SOR method shows that the novel approach of using a fixed-size block can produce a useful method. The four-point block SOR method is superior to the popular line SOR method for the model problem in respect of...
convergence, work needed per iteration, storage required and simplicity of programming. It does not converge as fast as the two-line SOR method, but is more efficient overall.

It is possible that larger fixed-size blocks would lead to even more effective methods. The principle might also be applied to non-cartesian geometries using, say, a four-point rhombic or rhomboidal block in skew co-ordinates, or a three-point triangular block in triangular co-ordinates.

Finally, the experimental results of Chapter 6 indicate the potential of approximate methods of pre-conditioned simultaneous iteration, but further work is required for them to be placed on a proper theoretical basis.
REFERENCES


INTERDATA (1973), "Interdata Users' Manual", (Publication 29-261R01), Interdata, Inc.


REID, J.K. (1966), "A method for finding the optimum successive over-relaxation parameter",

RUTHERFORD, D.E. (1945), "Some continuant determinants arising in physics and chemistry",

RUTHERFORD, D.E. (1951), "Some continuant determinants arising in physics and chemistry - II",
Prod. Royal Soc. Edinburgh, 63 (Section A).


STIEFEL, É. L. (1958), "Kernel polynomials in linear algebra and their numerical applications",

VARGA, R.S. (1962), "Matrix iterative analysis",


YOUNG, D.M. (1954), "Iterative methods for solving partial difference equations of elliptic type",

ICC Bulletin, 2, 1.

YOUNG, D.M. (1971), "Iterative solutions of large linear systems",
APPENDIX

Selected Program Listings
Four program listings are given here to illustrate the implementation of certain of the algorithms described in this thesis. These are:

1. Preconditioning with Miklin and Smolitsky's stencil, as described in section 4 of Chapter 2.

2. Peripheral successive over-relaxation in a triangular region, using the Gaussian elimination algorithm developed in sub-section 2.1 of Chapter 3.

3. Two-line successive over-relaxation in a parallelogram, using the Gaussian elimination algorithm presented in sub-section 3.1 of Chapter 3.

4. Four-point block successive over-relaxation as described in Chapter 5.
PRECONDITIONING WITH MIKHLIN AND SMOLITSKY'S STENCIL (CHAPTER 2)

1 'BEGIN'
2 'C'   SECOND-ORDER RICHARDSON METHOD 'C'
3 'C'
4   +
5   +  J-DIRECTION
6   +
7   +  NUMBERING 1 1 1
8   +  CONVENTION 1 1 1
9   +  >>>>>>>>
10  +  I-DIRECTION
11 'C'
12 'INT' N=5;
13 'INT' NL1=N-1;
14 [O:N,0:N1]REAL X;X1,T1;
15 'INT' ITN,I,J;
16 'REAL' ALPHA,RETA,S,W;
17 'REAL' EPS=5A-6;
18 'BOOL' CONVERGED;
19 'REAL' A=1,B=1,E=0.9,G=0;
20 [O:N]REAL C;
21
```
PROC MATPRINT=('INT' M,N,'REF'[,]'REAL' X):
  'BEGIN'
  'INT' NCOLM=8;
  'INT' GP=(N+1)/'NCOLM';
  'INT' N1=0,N2=NCOLM-1,NR=GP*NCOLM+1;
  'TO' GP 'DO'
    'BEGIN'
    PRINT((N1=0)NEWLINE+NEWPAGE));
    OUTF(STANDOUT,$51X"COLUMNS"<3>," TO"<3>$,(N1,N2));
    'FOR' I 'FROM' 0 'TO' M 'DO'
      OUTF( STANDOUT, $ LN( NCOLM ) (3X<1.582>)$ , (X[I,N1:N2]) ;
      N1 'PLUS' NCOLM; N2 'PLUS' NCOLM;
    'END'
  'IF' NR>0
  'THEN' N2 'PLUS' NR-NCOLM;
  OUTF( STANDOUT, $51X"COLUMNS"<3>," TO"<3>$,(N1,N2));
  'FOR' I 'FROM' 0 'TO' M 'DO'
    'BEGIN'
    FORMAT(STANDOUT,CLEAR FORMAT
       (SL4X7(JX<1.582>)$));
    OUTF( STANDOUT, X[I,N1:N2])
    'END'
  'FI'
  'END':
```
PRECONDITIONING WITH MIKHLIN AND SMOLITSKY'S STENCIL (CHAPTER 2)

46 'PROC' MULTAX=(REF[1],REAL) X.T):
47   'BEGIN'
48   'FOR' I 'TO' NL1 'DO'
49       'FOR' J 'TO' NL1 'DO'
50         'BEGIN'
52           T[I,J]=X[I,J]+S
53         'END'
54   'END'
55 'END';
56
57 'PROC' MULTWL=(REF[1],REAL) T):
58   'BEGIN'
59   'FOR' I 'TO' NL1 'DO'
60       'FOR' J 'TO' NL1 'DO'
61         'BEGIN'
63         'END'
64 'END';
65
66 'PROC' MULTWU=(REF[1],REAL) T):
67   'BEGIN'
68   'FOR' I 'FROM' NL1 'BY' -1 'TO' 1 'DO'
69       'FOR' J 'FROM' NL1 'BY' -1 'TO' 1 'DO'
70         'BEGIN'
71           T[I,J] 'MINUS' W*(C[1]*T[I+1,J]+C[2]*T[I,J+1]+C[8]*T[I+1,J-1])
72         'END'
73 'END';
'PROC' SETCOEFF='VOID';
'BEGIN'
'REAL' x=2*(A+B+E)+G;
'CLEAR' C1
C[1]=(A+E)/x;
C[2]=(R+E)/x;
C[3]=(A+F)/x;
C[4]=(R+F)/x;
C[6]=-F/x;
C[8]=-F/x;
C[0]=1
'END';

ALPHA=1.1407; BETA=0.3703; SETCOEFF;
'FOR' IW 'FROM' 0 'TO' 130 'DO';
'BEGIN'
W=0.01*IW;
'CLEAR' X; 'CLEAR' X1; 'CLEAR' T1;
'FOR' I 'TO' NL1 'DO';
'FOR' J 'TO' NL1 'DO'; X[I,J]=1;
CONVERGED='FALSE'; ITN=0;
PRECONDITIONING WITH MIKHLIN AND SMOLITSKY'S STENCIL (CHAPTER 2)

WHILE NOT CONVERGED DO
BEGIN
CONVERGED := 'TRUE'; ITN := ITN + 1
MULTA(X, Y); MULTWL(Y)
MULTWU(Y); FOR I 'TO' NL1 'DO'
FOR J 'TO' NL1 'DO'
BEGIN
S := BETA * X[I, J] - ALPHA * T[I, J];
('ABS' S > EPS | CONVERGED := 'FALSE');
X[I, J] := S;
X[I, J] := 'PLUS' S
END;
END;
OF ITERATION LOOP 'C'
OUTF(STDOUT, $L"RESULTS FOR W ="<1.2>," ALPHA ="<1.4>," BETA ="<1.4>$(W, ALPHA, BETA));
MATPRINT(N, N, X);
S := 0;
FOR I 'TO' NL1 'DO'
FOR J 'TO' NL1 'DO'
S := 'PLUS' X[I, J] * X[I, J];
OUTF(STDOUT, $L"EUCLIDEAN ERROR IN SOLUTION"<1.6>&2$, SQRT(S));
ALPHA := 1.3280; BETA := 0.0432
END;
OF W LOOP 'C'
END;
'FINISH'
***
TWO-LINE SORT IN A PARALLELOGRAM (CHAPTER 3)

```
'BEGIN'
'C'  >>> INCREASING J >>>
V
V
V
INCREASING I
V
V
V
DIRECTION OF SKEW
------
/
/
/
/
/

'C'
'INT' N=5; 'INT' NL1=N-1,NI2=N-2,TWONL1=2*N-2;
'REAL' H=1/N,K=1/N;
(0:1:N) 'REAL' X1;
(1:NL1) 'REAL' A3,B3,C3,D3,G3;
(1:TWONL1) 'REAL' DELTA,EPSILON,OMEGA,S;
[-1:TWONL1] 'REAL' ALPHABETA,GAMMA;
'REAL' ANGLE=PI/12;
'REAL' LAMBDASH=H*K,HSH=(H*SIN(ANGLE))**2,CA=COS(ANGLE);
'REAL' CP=2*(1+LAMBDASH);
'REAL' LAM2D=(LAMBDASH**2)/CP,OPD=LAMBDASH*CA/(2*CP),INV=1/CP,RHS=2*HS/CP;
'REAL' ERR=5*6;
'INT' ITN,1,J,1L1,1P1,1P2,L1;
'REAL' W,T:
'BOOL' CONVERGED;
'BOOL' ODDLINES=(N'/2)+2=N;
```
TWO-LINE SOR IN A PARALLELOGRAM (CHAPTER 3)

`PROC MATPRINT='INT M,N,'REF[,'REAL X];
  'BEGIN
    'INT NCOLM=8;
    'INT GP=(N+1)/NCOLM;
    'INT N1=0,N2=NCOLM-1,NR=N-GP+NCOLM+1;
    'TO GP 'DO'
      'BEGIN
        OUTF(standout,$2L51X"COLUMNS"<3>,"TO"<3>L$, (N1,N2));
        'FOR' I 'FROM' 0 'TO' M 'DO'
          OUTF(standout,:SL(NCOLM)(3X<1.5&2>),$X[I,N1:N2]);
        N1=N2+1;
        N2 'PLUS' NCOLM;
      'END';
    'IF' NR>0
      'THEN' OUTF(standout,$2L51X"COLUMNS"13>,"TO"13>L$, (N1,N2));
      'FOR' I 'FROM' 0 'TO' M 'DO'
        'BEGIN
          FORMAT(standout, CLEAR FORMAT
            ($L4X7(3X<1.5&2>)$));
          OUTF(standout,$X[I,N1:N2])
        'END'
      'FI';
  'END';`
TWO-LINE SOR IN A PARALLELOGRAM (CHAPTER 3)

`PROC` GAUTRISET=('INT' M,'REF[]'REAL' A3, B3, C3, D3, G3):
`BEGIN`
  D3[1]:=1/B3[1]; G3[1]:=C3[1]*D3[1];
  `FOR' I 'FROM' 2 'TO' M-1 'DO'
    `BEGIN`
      D3[I]:=1/(B3[I]-A3[I]*G3[I-1]);
      G3[I]:=C3[I]*D3[I];
    `END';
  `END';
`END';

`PROC` GAUTRICALC=('INT' M,'REF[]'REAL' A3, D3, G3, K[])'REAL';
`BEGIN'
  [1:M]'REAL' X;
  X[0]:=K[1] D3[1];
  `FOR' I 'FROM' 2 'TO' M 'DO'
    X[I]:= (K[I]+A3[I]*X[I-1])*D3[I];
  `FOR' I 'FROM' M-1 'BY' -1 'TO' 1 'DO'
    X[I] 'PLUS' G3[I]*X[I+1];
`END';
TWO-LINE SORT IN A PARALLELOGRAM (CHAPTER 3)

'PROC' GAUSEPTSET=('INT' M,'REF'(1)'REAL' ALPHA,BETA,GAMMA,DELTA, EPSILON,OMEGA)
BEGIN
'INT' M1=M-1, M2=M-2, M3=M-3
ALPHA[0]=ALPHA[-1]=0; BETA[0]=BETA[-1]=0; GAMMA[0]=GAMMA[-1]=0
GAMMA[1]=OPD*OMEGA[1];
FOR I FROM 2 TO M2 DO
'BEGIN'
'INT' IL3=I-3, IL2=I-2, IL1=I-1, IP1=I+1
DELTA[1]=INV*OPD*ALPHA[IL3];
EPSILON[1]=LAM2D+DELTA[1]*ALPHA[IL2]+OPD*BETA[IL2];
OMEGA[1]=1/(1-EPSILON[1]*ALPHA[IL1]-DELTA[1]*BETA[IL2])
-OPD*GAMMA[IL3];
ALPHA[1]=(-OPD+EPSILON[1]*BETA[IL1])*OMEGA[1];
BETA[1]=$(INV+EPSILON[1]*OMEGA[1])*OMEGA[1];
DELTA[IP1]=INV;
EPSILON[IP1]==-OPD+DELTA[IP1]*ALPHA[IL1];
OMEGA[IP1]=1/(1-EPSILON[IP1]*ALPHA[IP1]-DELTA[IP1]*BETA[IL1]);
ALPHA[IP1]=(LAM2D+EPSILON[IP1]*BETA[IP1]+DELTA[IP1]*GAMMA[IL1])
*OMEGA[IP1];
BETA[IP1]=INV*OMEGA[IP1];
GAMMA[IP1]=OPD*OMEGA[IP1];
'END';
BETA[M1]=0; GAMMA[M1]=0;
DELTA[M]=INV*OPD*ALPHA[M];
EPSILON[M]=LAM2D+DELTA[M]*ALPHA[M2]+OPD*BETA[M3];
OMEGA[M]=1/(1-EPSILON[M]*ALPHA[M1]-DELTA[M]*BETA[M2])
-OPD*GAMMA[M3];
ALPHA[M]=0; BETA[M]=0;
'END';
TWO-LINE SOLVE IN A PARALLELOGRAM (CHAPTER 3)

106 'PROC' GAUSSIANCALC=('[INT' M,'REF'[1]'REAL' ALPHA,BETA,GAMMA,DELTA,EPSILON,
107 OMEGA,K)[1]'REAL' ;
108 'BEGIN'
109 'INT' ML1=M-1,ML2=M-2,ML3=M-3; [1:M]'REAL' R,S;
113 'FOR' I 'FROM' 4 'BY' 2 'TO' MLZ 'DO'
114 'BEGIN'
115 'INT' IL3=I-3,IL2=I-2,IL1=I-1,IP1=I+1;
116 R[IL1]=(K[IL1]+EPSILON[IL1]*R[IL1]+DELTA[IL1]*R[IL2]+OPD*R[IL3])
117 +OMEGA[IL1];
118 R[IP1]=(K[IP1]+EPSILON[IP1]*R[1]+DELTA[IP1]*R[IL1])*OMEGA[IP1]
119 'END';
120 R[M]=R[M];
121 'END';
122 S[M]=R[M];
123 S[ML1]=R[ML1]+ALPHA[ML1]*S[M];
125 'FOR' I 'FROM' ML3 'BY' 2 'TO' 3 'DO'
126 'BEGIN'
127 'INT' IL1=I-1,IP1=I+1,IP2=I+2,IP3=I+3;
128 S[IL1]=R[IL1]+ALPHA[IL1]*S[IP1]+BETA[IL1]*S[IP2]+GAMMA[IL1]*S[IP3];
129 S[IL1]=R[IL1]+ALPHA[IL1]*S[IL1]+BETA[IL1]*S[IP1]
130 'END';
132 S
133 'END';
'PROC' W T AND A=('INT' I,J):

'BEGIN'

L1 'PLUS' 1

(S[L1] 'MINUS' X[I,J]) 'TIMES' W1

'IF' 'ARS' S[L1]>ARSERR

'THEN' CONVERGED='FALSE'

'FI'

X[I,J] 'PLUS' S[L1]

'END'

FOR' J 'FROM' 0 'TO' N 'DO' X[0,J]=X[N,J]=0;

FOR' I 'TO' N-1 'DO' X[I,0]=X[I,N]=0

GAUSEPTSET(TWO,ALPHA,BETA,GAMMA,DELTAEPSILON,OMEGA);

'IF' ODDLINES

'THEN' 'FOR' I 'TO' NL1 'DO'

'BEGIN'

A3[I]=C3[I]=INV; B3[I]=1

'END';

GAURISSET(NL1,A3,B3,C3,D3,G3)

'FI';
TWO-LINE FOR IN A PARALLELOGRAM (CHAPTER 3)

154 'FOR' I 'FROM' 100 'TO' 120 'DO'
155 'BEGIN'
156 \( W \leftarrow T W / 100; \)
157 'FOR' I 'TO' NL1 'DO'
158 'FOR' J 'TO' NL1 'DO' \( x[i,j] \leftarrow 1; \)
159 ITN:0 CONVERGED:='FALSE';
160 'WHILE' 'NOT' CONVERGED 'DO'
161 'BEGIN'
162 ITN 'PLUS' 1: CONVERGED:='TRUE';
163 'FOR' I 'BY' 2 'TO' NL2 'DO'
164 'BEGIN'
165 \( L1 \leftarrow I - 1; \)
166 \( IP1 \leftarrow I + 1; \)
167 \( IP2 \leftarrow I + 2; \)
168 L1:0;
169 'FOR' J 'TO' NL1 'DO'
170 'BEGIN'
171 \( L1 \leftarrow J - 1; \)
172 \( JP1 \leftarrow J + 1; \)
173 'END';
174 \( s[L1] \leftarrow RHS + OPD \times (x[1L1, JL1] - x[IL1, JP1]) + \lambda_2 \times x[1L1, JP1] + \lambda_3 \times x[JP2, JL1] + \lambda_4 \times x[IP2, J]; \)
176 \( L1 \leftarrow 0; \)
177 'FOR' J 'TO' NL1 'DO'
178 'BEGIN'
179 \( W T \leftarrow AND (A(i, j)); \)
180 \( W T \leftarrow AND (A(IP1, j)); \)
181 'END';
182 'END'; 'C' OF TWO-LINE LOOP 'C'
TWO-LINE SOR IN A PARALLELOGRAM (CHAPTER 3)

183 'IF' ODD_LINES
184 'THEN' I=NL1; IL1=I-1; IP1=I+1; L1=0;
185 'FOR' J 'TO' NL1 'DO'
186 'BEGIN'
187 'INT' JL1=J-1; JP1=J+1;
188 S[I,L1]=S[I,L1]+RHS*lam2d*(X[I,L1]*X[IP1,J])
189 -OPD*(X[I,L1]*X[IP1,JL1]-X[I,L1,LJ1]-X[IP1,JP1])
190 'END';
191 S[I,L1]=GAUSSCALC(NL1,A3,D3,G3,S);
192 L1=0;
193 'FOR' J 'TO' NL1 'DO' W T AND A(I,J)
194 'END'; 'C' OF CONVERGENCE LOOP 'C'
195 OUTF(StandOut,sp"CONVERGED AFTER"13>, " ITERATIONS. W ="11.2>",
196 (ITN,W));
197 MATHPRINT(N,N,X)
198 'END' 'C' OF IW LOOP 'C'
199 'END' 'C'
200 'FINISH'
201 ****
FOUR-POINT BLOCK SUCCESSIVE OVER-RELAXATION (CHAPTER 5)

\begin{verbatim}
1 'BEGIN'
2 'C' RED-BLACK ORDERING OF BLOCKS       'C'
3 'INT' N=5:
4 'INT' NL1=N-1, NL2=N-2:
5 [0:N,O:N] 'REAL' X;
6 'INT' IL1, JP1, JP2, JL1, JP1, JP2, ITN, K:
7 [1:4] 'REAL' SV, T:
8 'REAL' W, S:
9 'REAL' EPS=5E-6:
10 'BOOL' CONVERGED:
\end{verbatim}
FOUR-POINT BLOCK SUCCESSIVE OVER-RELAXATION (CHAPTER 5)

11 "PROC" MATPRINT="(INT M,N,REF[],)REAL X):
12  'BEGIN'
13  'INT' NCOLM=8;
14  'INT' GP=(N+1)/NCOLM;
15  'INT' N1=0,N2=NCOLM-1, NR=N-GP*NCOLM+1;
16  'TO' GP 'DO'
17   'BEGIN'
18   'IF' N1=0
19   'THEN' PRINT(NEWLINE);
20   'ELSE' PRINT(NEWPAGE);  
21   'FI':
22   OUTF(STANDOUT,$$1X"COLUMNS"!3>,$ TO"I3>($,(N1,N2));
23   'FOR' I FROM 0 TO M 'DO:
24     OUTF(STANDOUT,$LN(NCOLM)(3X<1.5&2>)$)+X[I,N1:N2]
25     N1 'PLUS' NCOLM: N2 'PLUS' NCOLM
26   'END';
27   'IF' NR>0
28     'THEN' N2 'PLUS' NR-NCOLM:
29     OUTF(STANDOUT,$$1X"COLUMNS"!3>,$ TO"<3>$,(N1,N2));
30   'FOR' I FROM 0 TO M 'DO:
31     'BEGIN'
32     FORMAT(STANDOUT,CLEAR FORMAT
33     ($L4X7(3X<1.5&2>)$));
34     OUTF(STANDOUT,X[I,N1:N2])
35     'END'
36   'FI:
37   'END";}
FOUR-POINT BLOCK SUCCESSIVE OVER-RELAXATION (CHAPTER 5)

38 'PROC' BLOCKEVAl.=('INT' I, J):
39 'BEGIN'
40 SX[1] = x[I, J] + x[I-1, J];
41 SX[2] = 2*(x[I-1, J-1] + x[I, J-1] + x[I+1, J] + x[I, J+1]);
42 SX[3] = x[1P1, J-1] + x[1P1, J+1];
46 T[21] = 0.25*(SX[4] + x[I+1, J-1] + x[I+1, J]);
47 T[41] = 0.25*(SX[4] + x[1P1, J] + x[1P1, J+1]);
48 'C' OVER-RELAXATION AND CONVERGENCE TEST 'C'
49 ((T[11] 'MINUS' x[I, J]) 'TIMES' W) 'PLUS' x[I, J]1
50 'IF' 'ARS' (T[11] - x[I, J]) > EPS 'THEN' CONVERGED = 'FALSE' 'FI';
51 x[I, J] = T[11];
52 ((T[21] 'MINUS' X[I, J-1]) 'TIMES' W) 'PLUS' x[I, J-1];
53 'IF' 'ARS' (T[21] - x[I, J-1]) > EPS 'THEN' CONVERGED = 'FALSE' 'FI';
54 x[I, J-1] = T[21];
55 ((T[31] 'MINUS' X[1P1, J-1]) 'TIMES' W) 'PLUS' x[1P1, J-1];
56 'IF' 'ARS' (T[31] - x[1P1, J-1]) > EPS 'THEN' CONVERGED = 'FALSE' 'FI';
57 x[1P1, J-1] = T[31];
58 ((T[41] 'MINUS' X[1P1, J]) 'TIMES' W) 'PLUS' x[1P1, J];
59 'IF' 'ARS' (T[41] - x[1P1, J]) > EPS 'THEN' CONVERGED = 'FALSE' 'FI';
60 x[1P1, J] = T[41];
61 'END';
FOR IW 'FROM' 100 'BY' 5 'TO' 195 'DO' 'BEGIN' W:=0.01*IW; 'FOR' J 'TO' NL1 'DO' 'FOR' I 'TO' NL1 'DO' X[I,J]:=1; 'FOR' J 'TO' NL1 'DO' X[I,J]:=X[I,N]; =0; 'FOR' J 'TO' NL1 'DO' X[0,J]:=X[N,J]; =0; CONVERGED;='FALSE'; ITN:=0; 'WHILE' 'NOT' CONVERGED 'DO' 'BEGIN' CONVERGED;='TRUE'; ITN='PLUS' 1; 'FOR' I 'BY' 2 'TO' NL2 'DO' 'BEGIN' JT=I-1; IP1:=I+1; IP2:=I+2; K=I+4+(I/4); 'FOR' J 'FROM' K 'BY' 4 'TO' NL2 'DO' 'BEGIN' JL1:=J-1; JP1:=J+1; JP2:=J+2; BLOCKVAL(I,J) 'END' 'C' OF J LOOP 'C' 'END' 'C' OF I LOOP 'C'
FOUR-POINT BLOCK SUCCESSIVE OVER-RELAXATION (CHAPTER 5)

FOR I 'BY' 2 'TO' NL2 'DO'
  BEGIN
    TL1 = I-1; IP1 = I+1; IP2 = I+2;
    K1 = I-4*(I+'4);
    (K=1) K=3 I K=1);
  FOR J 'FROM' K 'BY' 4 'TO' NL2 'DO'
    BEGIN
      JL1 = J-1; JP1 = J+1; JP2 = J+2;
      BLOKEVAL(I,J)
    ENN 'C' OF J LOOP 'C'
  END 'C' OF I LOOP 'C'
END 'C' OF CONVERGENCE LOOP 'C'
OUTF(STANDOUT, S5L "W ="<1.2>," CONVERGED IN"<4>" ITERATIONS"$(W,ITN))
MATPRINT(N,N,X)
END 'C' OF W LOOP 'C'
FINISH
***
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

'BEGIN'
'INT' P=30;
'INT' NPERI=P/3, MAXSIZE=3*(P-3);
[1;MAXSIZE] REAL' S;
[1,NPERI-1] REF[1]' REAL' A,C,D,E, DEN,G,H, CAPG, T, U;
'REAL' STEP=1/P;
'REAL' RHS=(3*STEPH*STEPH)/6;
'INT' LIMIT, C, TNL, I, IL1, IP1, J, JL1, JP1, SIZE, SL1, SP1;
'REAL' SIXTH=1/6, EPS=5E-6;
'REAL' T1,W;
'BOOL' CONVERGED;
'MODE' TRIMAT=[0:P]'REF[1]'REAL';
TRIMAT X;

PROC TRIMATPRINT=(REF TRIMAT Y):
'BEGIN'
'INT' M='UPB' Y[0];
'FORMAT' TM='SL6(3X<1.882>)$';
'FOR' I 'FROM' 0 'TO' M 'DO'
'REGIN'
OUT(STANDOUT, $L"LINE"<3>$, I))
FORMAT(STANDOUT, CLEAR FORMAT(TM));
OUT(STANDOUT, Y[1])
'FND'
'END';
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

`PROC' GAUSETCOEFF=('REF[],'REAL,'A,C,D,E,DEN,G,H,CAPG,T,U);
`BEGIN' 'C' NEGATIVES OF OFF-DIAGONAL COEFFICIENTS 'C'
`INT' N=UPB' A-1; 'INT' NL1=N-1; 'REAL' CAPH;
`INT' P=N/'3; 'INT' TP=2*P;
`PROC' MAINS=('INT' K);
`BEGIN'
`INT' KL1=K-1;
DEN[K]:=1/(1-A(K)*G(KL1));
G[K]:=C(K)*DEN[K];
H[K]:=A(K)*H(KL1)*DEN[K];
CAPH 'MINUS' CADG(KL1)*H(KL1); CAPH[K]:=CAPG(KL1)*G(KL1);
END';
`PROC' SUBS=('INT' K);
`BEGIN'
`INT' KL1=K-1,KP1=K+1,J=K*/P;
UFJ1:=E(J)*DEN(KL1);
DEN[K]:=1/(1-A(K)*G(KL1));
G[K]:=C(K)+A(K)*U(J)+DEN[K];
H[K]:=A(K)*H(KL1)*DEN[K];
CAPH 'MINUS' CADG(KL1)*H(KL1); CAPH[K]:=CAPG(KL1)*G(KL1);
TJ1:=A(KP1)+D(J)*G(KL1);
DEN[KP1]:=1/(1-D(J)*U(J)-T(J)*G(K));
G[KP1]:=C(KP1)+DEN[KP1];
H[KP1]:=D(J)+H(KL1)+T(J)*H(K)+DEN[KP1];
CAPH 'MINUS' CADG(K)*H(K); CAPH[K]:=CAPG(KP1)*G(K)+CAPG(KL1)*UFJ;
'FND';
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

56  CAPH := 1;  CAPG[1] := C[N];
60  'FOR' I 'FROM' 3 'TO' P 'DO' MAINS(I);
61  SUBS(P+1);
62  'FOR' I 'FROM' P+3 'TO' TWOP 'DO' MAINS(I);
63  SUBS(TWOP+1);
64  'FOR' I 'FROM' TWOP+3 'TO' NL1 'DO' MAINS(I);
65  CAPH 'MINUS' (A[N] + CAPG[NL1])*(G[NL1] + H[NL1]);
66  DEN[N] := 1/CAPH
67  'END';
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

68 'PROC' GAUCALC=('RELF'[ ]'REAL' A,D,DEN,G,H,T,U,CAPG,K)[]'REAL'1
69 'BEGIN'
70 'INT' N=1: 'INT' NL1=N-1; [1;N]'REAL' F; 'REAL' CAPF;
71 'INT' P=N/'3'; 'INT' TWOP=N-P;
72 'PROC' MAINC=('INT' J):
73 'BEGIN'
74 'INT' JL1=J-1;
75 F(J1)=A(J1)*F(JL1)*DEN(J);
76 'END';
78 'PROC' SUBC=('INT' J):
79 'BEGIN'
80 'INT' JL1=J-1,JL2=J-2,K1=J/'P';
81 F(J1)=A(J1)*F(JL1)*T(K1)*F(JL2)*F(JL1)*DEN(J);
82 'END';
84 'PROC' MAINBACK=('INT' J):
85 'BEGIN'
86 F(J1)='PLUS' G[J1+F[J+1]+H[J]*F[N]
87 'END';
88 'PROC' SUBBACK=('INT' J):
89 'BEGIN'
90 MAINBACK(J);
91 F(J1)='PLUS' U[J1/'P]*F[J+2]
92 'END';
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

93 \[ F(11) = K(11); \text{CAPF} = K(N); \]
94 'FOR' I 'FROM' 2 'TO' P+1 'DO' MAINC(I); SUBC(P+2);
95 'FOR' I 'FROM' P+3 'TO' TWOP+1 'DO' MAINC(I); SUBC(TWOP+2);
96 'FOR' I 'FROM' TWOP+3 'TO' NL1 'DO' MAINC(I);
97 \[ F(N) = (\text{CAPF}(A[N]) + \text{CAPG}(NL1)) + F(NL1) \times \text{DEN}(N); \]
98 'FOR' I 'FROM' NL1 'BY' -1 'TO' TWOP+1 'DO' MAINBACK(I);
99 SUBBACK(TWOP);
100 'FOR' I 'FROM' TWOP-1 'BY' -1 'TO' P+1 'DO' MAINBACK(I);
101 SUBBACK(P);
102 'FOR' I 'FROM' P-1 'BY' -1 'TO' 1 'DO' MAINBACK(I);
103 F
104 'END';
105 'PROC' W T AND A=('INT' I,J):
106 'BEGIN'
107 L1 'PLUS' 1;
108 (S[L1] 'MINUS' X[I][J]) 'TIMES' W;
109 'IF' 'ARS' S[L1] > EPS
110 'THEN' CONVERGED = 'FALSE'
111 'FI';
112 X[I][J] 'PLUS' S[L1]
113 'END';
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

114 IC:= 0;
115 DECLARE:
116 X[IC]:= 'LOC'[0:P-IC]'REAL';
117 'IF' (IC'PLUS' 1)<P
118 'THEN' 'GOTO' DECLARE
119 'FI';
120 IC:= 1;
121 DEC2:
122 SIZE:=(NPERI-IC)+9; SL1:=SIZE-1; SP1:=SIZE+1;
123 A[IC]:= 'LOC'[1:SP1]'REAL'; 'FOR' I 'TO' SP1 'DO' A[IC][I]:=SIXTH;
124 C[IC]:= 'LOC'[1:SP1]'REAL'; 'FOR' I 'TO' SP1 'DO' C[IC][I]:=SIXTH;
125 D[IC]:= 'LOC'[1:2]'REAL'; 'FOR' I 'TO' 2 'DO' D[IC][I]:=SIXTH;
126 E[IC]:= 'LOC'[1:2]'REAL'; 'FOR' I 'TO' 2 'DO' E[IC][I]:=SIXTH;
127 DEN[IC]:= 'LOC'[1:SIZE]'REAL';
128 G[IC]:= 'LOC'[1:SL1]'REAL';
129 H[IC]:= 'LOC'[1:SL1]'REAL';
130 CAPG[IC]:= 'LOC'[1:SL1]'REAL';
131 T[IC]:= 'LOC'[1:2]'REAL';
132 U[IC]:= 'LOC'[1:2]'REAL';
133 'IF' (IC'PLUS' 1)<NPERI
134 'THEN' 'GOTO' DEC2
135 'FI';
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

136 'FOR' I 'TO' NPERI-1 'DO'
137    GAUSETCOEFF(A[I],C[I],D[I],E[I],DEN[I],G[I],H[I],CAPG[I],T[I],U[I]);
138 'C' 'SET BOUNDARY CONDITIONS' 'C'
139 'FOR' I 'FROM' 0 'TO' P 'DO' X[I][0] := 0;
140 'FOR' J 'FROM' 0 'TO' P 'DO' X[0][J] := 0;
141 I := 0; J := P;
142 'WHILE' J > 1 'DO'
143    'BEGIN'
144        I 'PLUS' 1; J 'MINUS' 1;
145        X[I][J] := 0
146    'END';
147 'END';
148 'FOR' I 'FROM' 164 'TO' 168 'DO'
149    'BEGIN'
150        W := IW/100;
151 'C' 'SET FIRST APPROXIMATION' 'C'
152    'FOR' I 'TO' P-2 'DO'
153        'FOR' J 'TO' P-1-1 'DO' X[I][J] := 1;
154    CONVERGED := 'FALSE'; YTN := 0;
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

'WHILE' 'NOT' CONVERGED 'DO'
'BEGIN'
CONVERGED := 'TRUE'; ITN 'PLUS' 1;
'FOR' KP 'TO' NPERI-1 'DO'
'BEGIN'
'INT' SIDELENGTH = P - 3 * KP;
'INT' TRISIZE = 3 * SIDELENGTH;
'SSIDE = 1' 'C'
I := KP; IL1 := I - 1; IP1 := I + 1;
J := KP; JL1 := J - 1; JP1 := J + 1;
S[I] := RHS + (X[I][JL1] + X[IP1][JL1] + X[IL1][J] + X[IL1][JP1]) * SIXTH;
SL1 := I; I := IP1; IP1 'PLUS' 1;
S[I] := RHS + (X[I][JL1] + X[IP1][JL1] + X[I][JP1]) * SIXTH;
L1 := 1;
'TO' SIDELENGTH - 3 'DO'
'BEGIN'
IL1 := I; I := IP1; IP1 'PLUS' 1; L1 'PLUS' 1;
S[IL1] := RHS + (X[I][JL1] + X[IP1][JL1] + X[I][JP1] + X[IL1][JP1]) * SIXTH;
'END';
'I1 := I; I := IP1; IP1 'PLUS' 1; L1 'PLUS' 1;
S[IL1] := RHS + (X[I][JL1] + X[IP1][JL1] + X[IL1][JP1]) * SIXTH;
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

```
177  'C' SIDE = 7
178  'C'
179  II1:=I;  I:=IP1;  IP1 'PLUS' 1;  L1 'PLUS' 1;
180  S[I][I]:=RHS+(X[IP1][J]+X[I][JP1]+X(IP1)[JL1]+X[I][JL1])*SIXTH;
181  I1:=I;  I:=IL1;  IL1 'MINUS' 1;
182  JL1:=J;  J:=JP1;  JP1 'PLUS' 1;  L1 'PLUS' 1;
183  S[I][I]:=RHS+(X[IP1][J]+X[I][JP1]+X[IL1][J])*SIXTH;
184  'TO' SIDELENGTH=3 'DO'
185  'BEGIN'
186          IP1:=I;  I:=IL1;  IL1 'MINUS' 1;
187          JL1:=J;  J:=JP1;  JP1 'PLUS' 1;  L1 'PLUS' 1;
188          S[I][I]:=RHS+(X[IP1][J]+X[I][JP1]+X[I][JL1]+X[IL1][J])*SIXTH;
189  'END';
190  IP1:=I;  I:=IL1;  IL1 'MINUS' 1;
191  JL1:=J;  J:=JP1;  JP1 'PLUS' 1;  L1 'PLUS' 1;
192  S[I][I]:=RHS+(X[IP1][J]+X[I][JP1]+X[I][JL1]+X[IL1][J])*SIXTH;
193  I1:=I;  I:=IL1;  IL1 'MINUS' 1;
194  JL1:=J;  J:=JP1;  JP1 'PLUS' 1;  L1 'PLUS' 1;
195  S[I][I]:=RHS+(X[IP1][J]+X[I][JP1]+X[IL1][JP1]+X[IL1][J])*SIXTH;
```
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

\[ \text{SIDE = 3} \]

\[ \text{JP}1 = \text{JP}1; \text{J} = \text{JL}1; \text{JL}1 \text{ 'MINUS' 1}; \text{L}1 \text{ 'PLUS' 1}; \]
\[ \text{SF1L1} = \text{RHS} \times (X[I][L][J] + X[I][L][JP1] + X[I][L][JL1]) \times \text{SIXTH}; \]
\[ \text{'TO' SIDE LENGTH - 3 'DO'} \]
\[ \text{'BEGIN'} \]
\[ \text{JP}1 = \text{JP}1; \text{J} = \text{JL}1; \text{JL}1 \text{ 'MINUS' 1}; \text{L}1 \text{ 'PLUS' 1}; \]
\[ \text{SF1L1} = \text{RHS} \times (X[I][L][J] + X[I][L][JP1] + X[I][JL1] + X[I][JIP1]); \]
\[ \times \text{SIXTH}; \]
\[ \text{'END'} ; \]
\[ \text{JP}1 = \text{JP}1; \text{J} = \text{JL}1; \text{JL}1 \text{ 'MINUS' 1}; \text{L}1 \text{ 'PLUS' 1}; \]
\[ \text{SF1L1} = \text{RHS} \times (X[I][L][J] + X[I][L][JP1] + X[I][JL1] + X[I][JIP1]); \]
\[ \times \text{SIXTH}; \]
\[ \text{'C' SOLVE EQUATIONS AND TEST FOR CONVERGENCE 'C'} \]
\[ \text{SF1[TRIS17E]} \] = GAUCALC
\[ (A[KP], D[KP], DEN[KP], G[KP], W[KP], T[KP], U[KP], CAPG[KP], S); \]
\[ \text{L1} = 0; \]
\[ \text{LIMIT} = \text{KP} - 2 \times \text{KP}; \]
\[ 'FOR' \text{ J 'FROM' KP 'TO' LIMIT 'DO' W T AND A(I,KP)} ; \]
\[ \text{I} = \text{LIMIT}; \text{J} = \text{KP}; \]
\[ 'WHILE' \text{ I>}KP 'DO' W T AND A(I 'MINUS' 1, J 'PLUS' 1); \]
\[ 'FOR' \text{ J 'FROM' LIMIT 'BY' -1 'TO' KP + 1 'DO' W T AND A(KP, J)} ; \]
\[ 'END'; 'C' OF PERIPHERAL LOOP 'C'. \]
PERIPHERAL SOR IN A TRIANGULAR REGION (CHAPTER 3).

217 'C' CENTRF POINT 'C'
218 I:=NPERI; IL1:=I-1; JP1:=I+1;
219 J:=I; JL1:=IL1; JP1:=IP1;
220 T1:=RHS+(X[I][JL1]+X[I][JP1]+X[IL1][J]+X[IP1][J]
221 +X[IL1][JP1]+X[IP1][JL1])*SIXTH;
222 IF 'MINUS' X[I][J] 'TIMES' W
223 THEN CONVERGED:='FALSE'
224 IF 'ABS' T1>EPS
225 THEN OF CONVERGENCE LOOP 'C'
226 X[I][J] 'PLUS' T1
227 'END': 'C' OF CONVERGENCE LOOP 'C'
228 OUTF(StanDOut.$P"W = "<1.2>," CONVERGED IN"<4>" ITERATIONS$,(W,ITN));
229 TRIMAPRINT(X)
230 'END': 'C' OF W LOOP 'C'
231 'END'
232 'FINISH'
233 ***