Block Iterative Methods for Three-Dimensional Groundwater Flow Models

K.J.Neylon

September 1991

Submitted to the

Department of Mathematics,

University of Reading,

in partial fulfilment of the requirements for the

Degree of Master of Science

Abstract

Some block iterative methods which are applied to the matrices resulting from the finite element approximation of a three dimensional non-linear partial differential equation are compared. The nodal ordering is shown to have an important important effect on the convergence of these methods. The possibility of implementing the methods on a transputer system is discussed.

Acknowledgements

I would like to thank Dr.M.J.Baines and Dr.N.K.Nicholls, of the Mathematics Department, University of Reading, for their help, support and advice during this work.

I would also like to acknowledge the financial support of the SERC.

Contents

List of Figures

List of Tables

Notation

Symbol	**Meaning**
e_j	unit gravitational vector
h	size of element as proportion of whole region
H	size of rectangular prism element in the z-direction
k	iteration number
k_{rw}	relative permeability with respect to the water phase
K_{ij}	saturated hydraulic conductivity tensor
l	size of rectangular prism element in the x-direction
N_J	three dimensional linear basis function
m	size of rectangular prism element in the y-direction
p	number of processors in transputer network
s	number of diagonal blocks in the matrix system
S_w	water saturation
x_1, x_2, x_3	Cartesian co-ordinate directions
$x\ ,\ y\ ,\ z$	Cartesian co-ordinate directions
ψ	pressure head
ψ_J	approximate nodal pressure head on the finite element grid
$\rho(M)$	spectral radius of matrix M
ω	over-relaxation parameter
$\mathcal{R}$	set of real numbers

Chapter 1

Introduction

In general, there are two approaches when solving matrix problems - direct and iterative. Direct methods solve the problem in a known (finite) number of operations and errors in the solution arise entirely from rounding errors introduced in the computation. Iterative methods generate a sequence of approximate solutions (iterates) which tend towards the solution of the problem; these methods usually exploit any sparsity in the matrix more than direct methods.

This dissertation is concerned with block iterative methods for the solution of large sparse systems of equations arising from a three dimensional finite element model of groundwater flow, consideration is given to the possibility of implementing these algorithms on a parallel computer. The investigations carried out can be divided into three main areas.

The first area of investigation is the choice of node numbering scheme. At present, many three dimensional finite element models use a single node numbering scheme over the whole region, chosen without any consideration given to the predominant direction of flow in the region (e.g. the vertical slicing approach employed in [9]). It is conjectured that the block diagonal dominance governs the asymptotic rate of convergence of block iterative methods so the effect of the orientation of the node numbering scheme with respect to the predominant direction of flow on the block diagonal dominance is examined.

The second area of investigation is the performance of some block matrix iterative methods on the type of matrix generated in the first part of the dissertation. The final area of investigation is the possibility of implementation of the block iterative methods in a parallel computing architecture. The target machine is a transputer system which is currently at Reading University on loan from the Rutherford Appleton Laboratory as part of the D.T.I./S.E.R.C. Initiative in the Engineering Application of Transputers. The standard block algorithms will be presented in such a way as to be executable on this type of system.

In Chapter 2, the governing equation for groundwater flow is presented, the finite element approximation is described and a sample problem is given. In Chapter 3, some background matrix theory and the sequential iterative block matrix methods are described. In Chapter 4, the effect of different nodal ordering approaches is investigated for sample matrix problems generated by the techniques in Chapter 2. The performance of the block matrix methods described in Chapter 3 is examined for some of the sample problems in Chapter 5, and in Chapter 6 these block iterative methods are directed at a transputer system. Finally, in Chapter 7 closing comments are made and some conclusions are drawn about nodal ordering and block iterative solution methods.

Chapter 2

Generation of Matrix Systems

The matrix systems used during the course of this project are generated by finite element approximations to simple three dimensional groundwater flow problems. The approach used is based on the work outlined in [8, 9]. A description of the process of matrix generation is given in this chapter and a sample problem (which is used throughout the dissertation) is presented.

2.1 Governing Equation

From [8] the governing equation for three dimensional flow in a variably saturated porous medium is

$$\frac{\partial}{\partial x_i}\left[K_{ij}k_{rw}\left(\frac{\partial\psi}{\partial x_j}+e_j\right)\right]=\left(S_wS_s+\phi\frac{dS_w}{d\psi}\right)\frac{\partial\psi}{\partial t}-q \qquad i,j=1,2,3 \tag{2.1}$$

where	ψ	is the pressure head
	K_{ij}	is the saturated hydraulic conductivity tensor
	k_{rw}	is the relative permeability with respect to the water phase
	e_j	is the unit gravitational vector
	q	is the volume flow rate via sources (or sinks)
	S_w	is the water saturation

S_s is the specific storage coefficient

ϕ is the porosity

t is time

and x_1 , x_2 , x_3 are Cartesian co-ordinate directions.

In order to simplify this equation, the range of problems considered is restricted to those which are in steady state (i.e. no time dependence) and have no sources or sinks. Under these conditions (2.1) becomes

$$\frac{\partial}{\partial x_i}\left[K_{ij}k_{rw}\left(\frac{\partial\psi}{\partial x_j}+e_j\right)\right]=0 \qquad i,j=1,2,3 \tag{2.2}$$

This equation is non-linear since both the relative permeability and the saturated hydraulic conductivity tensor are functions of the solution (the pressure head) in the unsaturated part of the region [4].

2.2 Finite Element Approximation

The standard Galerkin method [2, 12] is used to generate the three dimensional finite element approximation to (2.2). A trial function, $\tilde{\psi}$, approximating the unknown, ψ, is selected of the form

$$\tilde{\psi}(x_i)=\sum_{J=1}^{n}\psi_J N_J(x_i) \qquad i=1,2,3$$

where ψ_J is the approximate nodal value of $\tilde{\psi}$ on the finite element grid

N_J is a three dimensional linear basis function

n is the number of nodes in the finite element grid

The partial differential equation (2.2) is multiplied by a test function, w, where

$$w=N_I(x_i) \qquad i=1,2,3 \qquad I=1,2,...,n$$

is the same basis function as for the trial space, and then integrated over the whole region. Green's theorem is applied to transform the second derivative term (which is not defined for linear basis functions) to a first derivative term (which is defined inside each element) and finally, ψ is replaced by $\tilde{\psi}$. The end result of the finite element approximation is a system of equations,

$$\sum_J A_{IJ}\psi_J = F_I \qquad I = 1, 2, ..., n \tag{2.3}$$

where

$$A_{IJ} = \sum_e \int_{R^e} K_{ij} k_{rw} \frac{\partial N_I}{\partial x_i} \frac{\partial N_J}{\partial x_j} dR \tag{2.4}$$

$$F_I = -\sum_e \int_{R^e} K_{ij} k_{rw} \frac{\partial N_I}{\partial x_i} e_j dR \tag{2.5}$$

where A_{IJ} is the (I, J) th entry in the global seepage matrix

F_I is the I th entry in the global right hand side vector

and the summations are performed over all the elements in the region. Note that since A_{IJ} is a function of ψ_J then (2.3) is non-linear. From [12], the error between the approximate solution given by the finite element method with linear basis functions and the exact solution to the problem is $0(h^2)$ where h represents the size of the largest element. e.g.

$$h = \frac{\text{Size of Largest Element}}{\text{Size of Region}}$$

So the finite element method will give an approximate solution which converges to the analytic solution as $h \to 0$ (i.e. the method is consistent).

In practice, the global matrices and vectors in a finite element program are generated by assembling the element matrices and vectors. The positions of the entries in the element matrices correspond to the local node numbering within an element (see Figure 2.1). For example, the $(i_{local}, j_{local})^{th}$ entry in an element matrix represents the connection between local node i_{local} and local node j_{local} in the discrete approximation of the differential equation. In order to place an entry from an element matrix in the corresponding position in the global matrix, the local node numbering is converted into global node numbering.

$$\text{i.e.} \qquad (i_{local}, j_{local}) \Rightarrow (i_{global}, j_{global})$$

Since a node at position (i_{global}, j_{global}) will be a part of more than one element, the contribution from each element must be combined to give the global matrix entry.

2.2.1 Computation of Element Matrices

Only rectangular prism elements are used in the discretisation of the region - hence only rectangular regions can be modelled. These elements are chosen so that they are all of the same size and are aligned with the global Cartesian coordinates (with the y-direction taken as vertically upwards). A typical element is shown in Figure 2.1. The element seepage matrices, $[A]^e$, and element right

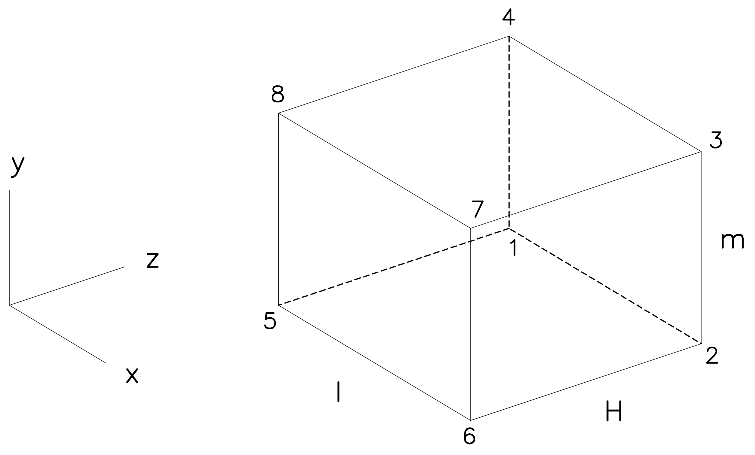

Figure 2.1: Dimensions and Orientation of Elements

hand side vectors, $\{F\}^e$, are generated using the influence coefficient technique described in [9]. This technique uses the fact that the elements are simple and regular to calculate the coefficients in the element matrices and vectors for a general element. It cannot be used effectively on regions with more widely varying types of element geometry where numerical quadrature must be used to generate the coefficients for each element independently.

The expressions for the seepage matrix and right hand side vectors (2.4),(2.5) are simplified by taking the hydraulic properties to be constant within each element (and equal to the centroidal value). This corresponds to one-point three-dimensional Gaussian quadrature on a rectangular region which introduces an error of $0(h^2)$ i.e. the same order of accuracy as the finite element approximation. The element seepage matrix is given by

$$\begin{aligned}[A]^e &= \langle K_{xx}k_{rw}\rangle \frac{mH}{2l}[A^{xx}]^e + \langle K_{yy}k_{rw}\rangle \frac{lH}{2m}[A^{yy}]^e \\ &+ \langle K_{zz}k_{rw}\rangle \frac{lm}{2H}[A^{zz}]^e + \langle K_{xy}k_{rw}\rangle \frac{H}{2}[A^{xy}]^e \\ &+ \langle K_{yz}k_{rw}\rangle \frac{l}{2}[A^{yz}]^e + \langle K_{zx}k_{rw}\rangle \frac{m}{2}[A^{zx}]^e\end{aligned}$$

where $\langle\cdot\rangle$ denotes the centroidal value and, because the global coordinate system is oriented such that y is vertically upwards, then the element right hand side vector is given by

$$\{F\}^e = -\langle k_{rw}\rangle[\langle K_{xy}\rangle\{F^x\}^e + \langle K_{yy}\rangle\{F^y\}^e + \langle K_{zy}\rangle\{F^z\}^e],$$

The influence coefficient matrices and vectors are as given in Appendix A.

2.3 Sample Problem

The problem used to examine the nodal ordering and block iterative methods in this dissertation is a modified form of an unsaturated flow example given in [3]. This example is essentially a two-dimensional problem, to be solved by a three-dimensional finite-element package by keeping all the characteristics of the problem constant across the width of the region and using a single element across the width.

Boundary	Condition
$x = 0 \ cm$	$\psi = y(1+z) \ cm \quad (y > 6)$ No flow $(y \leq 6)$
$x = 15$ cm	$\psi = -180$ cm
$y = 0$ cm	No flow
$y = 10$ cm	No flow
$z = 0$ cm	No flow
$z = 10$ cm	No flow

Table 2.1: Boundary Conditions for Sample Problem

In order to make this problem fully three-dimensional, one of the boundary conditions is altered from being constant across the width to one in which the prescribed pressure head is dependent on the position across the width (i.e. in the z-direction). The geometry and boundary conditions for the sample problem are shown in Figure 2.2 and Table 2.1 respectively.

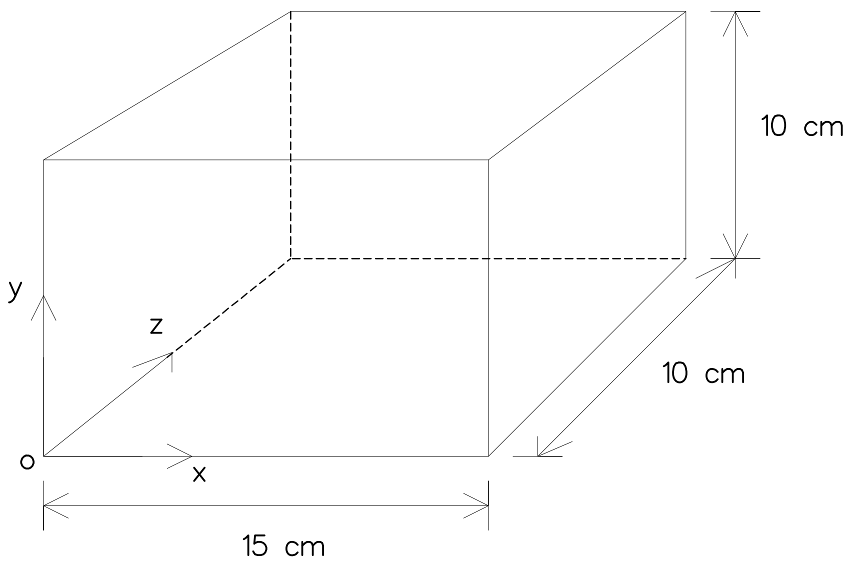

Figure 2.2: Geometry of Sample Problem

The following constitutive relationships are used for the soil moisture characteristics,

$$k_{rw} = \frac{S_w - S_{wr}}{1 - S_{wr}} \qquad , \qquad \frac{1 - S_w}{1 - S_{wr}} = \frac{\psi_a - \psi}{\psi_a - \psi_r}$$

where ψ_a is the air pressure head entry value

ψ_r is the residual pressure head

S_{wr} is the residual water saturation

and $\psi_a = 0$ cm , $\psi_r = -200$ cm , $S_{wr} = 0.333$

The solution on the top surface of the region (i.e. at $y = 10$ cm) with nine elements in each direction is shown in Figure 2.3.

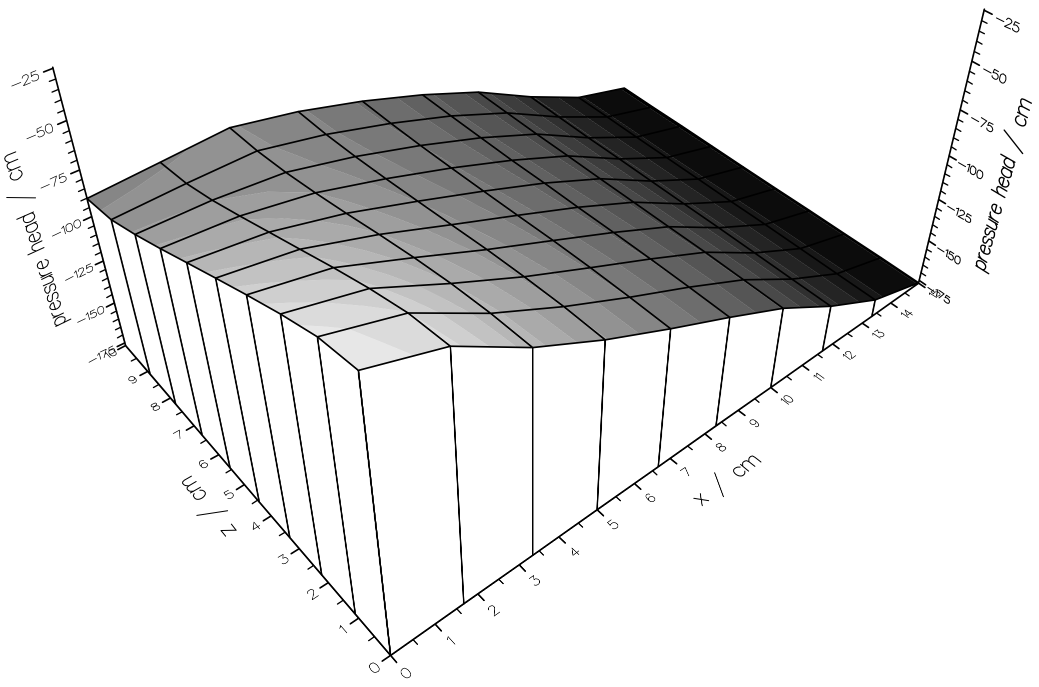

Figure 2.3: Pressure Head Distribution on Top Surface of Region

Chapter 3

Matrix Theory and Sequential Algorithms

In this chapter, the block matrix theory used during the dissertation is outlined. Some standard sequential iterative block matrix algorithms are given and the performance of these is compared for some sample problems.

Where appropriate, the theory is directed at a symmetric, block tri-diagonal matrix, since this is the type of matrix generated by the finite element approximation described in Chapter 2 in conjunction with the slicing discretisation approaches described later in this dissertation. An example of a system involving this type of matrix, with $s \times s$ blocks, is shown in (3.1).

$$\begin{pmatrix} A_1 & B_1 & & & \\ B_1^T & A_2 & B_2 & & \\ & \ddots & \ddots & \ddots & \\ & & B_{s-2}^T & A_{s-1} & B_{s-1} \\ & & & B_{s-1}^T & A_s \end{pmatrix} \begin{pmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_{s-1} \\ \underline{x}_s \end{pmatrix} = \begin{pmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_{s-1} \\ \underline{b}_s \end{pmatrix} \tag{3.1}$$

3.1 Background Theory and Definitions

Spectral Radius

In general, if A is an $n \times n$ complex matrix with eigenvalues, $\lambda_i, \quad i = 1, \ldots n$ then

$$\rho(A) = \max_{1 \leq i \leq n} |\lambda_i|$$

is the spectral radius of the matrix A.

Asymptotic Rate of Convergence [13]

The asymptotic rate of convergence of a linear stationary iterative method (such as those described in Section 3.2) with iteration matrix A, $R_\infty(A)$, is

$$R_\infty(A) = -\ln\{\rho(A)\}$$

It is a measure of the rapidity of convergence of the iterative method.

Condition Number

For a square matrix, A, the condition number, $\kappa(A)$, is

$$\kappa(A) = \parallel A \parallel \parallel A^{-1} \parallel$$

with the convention that $\kappa(A) = \infty$ for singular A. The condition number quantifies the sensitivity of the $A\underline{x} = \underline{b}$ problem to numerical solution, the larger the condition number the more sensitive the problem to numerical inversion. Note that $\kappa(\cdot)$ depends on the underlying norm, when this norm is to be stressed subscripts are used, i.e.

$$\kappa_2(A) = \parallel A \parallel_2 \parallel A^{-1} \parallel_2$$

if the underlying matrix norm is the 2-norm.

Consistent Ordering [6]

The $s \times s$ block matrix, A, consisting of blocks $A_{i,j}$ $(i, j = 1, \ldots, s)$ is consistently ordered if for some t there exist disjoint non-empty subsets $S_1, \ldots, S_t$ of $\{1, 2, \ldots, s\}$ such that

$$\bigcup_{i=1}^{t} S_i = \{1, 2, \ldots, s\}$$

and such that if $A_{i,j} \neq 0$ with $i \neq j$ and S_k is the subset containing i, then $j \in S_{k+1}$ if $j > i$, and $j \in S_{k-1}$ if $j < i$.

p-cyclic Matrix [13]

An $n \times n$ complex matrix A is weakly cyclic of index k (> 1) if there exists an $n \times n$ permutation matrix, P, such that PAP^T is of the form

$$PAP^T = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & A_{1,k} \\ A_{2,1} & 0 & & & 0 & 0 \\ 0 & A_{3,2} & \ddots & & & 0 \\ \vdots & \vdots & \ddots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & & A_{k,k-1} & 0 \end{pmatrix}$$

If the block Jacobi matrix of A (Section 3.2.2) is weakly cyclic of index p (≥ 2) then A is a p-cyclic matrix. It can be shown that a block tri-diagonal matrix is a consistently ordered 2-cyclic matrix [13].

Block Diagonal Dominance

An $s \times s$ block matrix, A, consisting of blocks $A_{i,j}$ $(i, j = 1, \ldots, s)$ is block diagonally dominant [5] if, for $i = 1, \ldots, s$

$$\|A_{i,i}^{-1}\|_\infty \sum_{\substack{j=1 \\ j \neq i}}^{s} \|A_{i,j}\|_\infty \leq 1$$

This definition can be modified to give an expression for the *degree of block diagonal dominance*, σ, of a block tridiagonal matrix as follows.

$$\sigma = \max_{i=1,.,s} \left\{ \|A_{i,i}^{-1}\|_\infty (\|A_{i,i-1}\|_\infty + \|A_{i,i+1}\|_\infty) \right\} \qquad A_{1,0} = A_{s,s+1} = 0$$

The smaller the value of σ, the stronger the block diagonal dominance.

3.2 Classical Iterative Methods for Block Matrices

The classical iterative techniques [13] for solving the linear system,

$$A\underline{x} = \underline{b} \qquad \underline{b} \in \mathcal{R}^n, A \in \mathcal{R}^{n \times n} \tag{3.2}$$

are based on splittings of the matrix, A. The standard notation for a splitting is

$$A = M - N \qquad (M \quad \text{invertible}) \tag{3.3}$$

giving rise to the iterative method

$$M\underline{x}^{k+1} = N\underline{x}^k + \underline{b} \qquad k \geq 0 \tag{3.4}$$

where $\underline{x}^{k+1}$ is the next iterate based on the previous iterate, $\underline{x}^k$.
$M^{-1}N$ is the iteration matrix and it can be shown [5] that $\rho(M^{-1}N) < 1$ is a necessary and sufficient condition for the method to converge to the true solution, $\underline{x} = A^{-1}\underline{b}$, for any initial iterate, $\underline{x}^0$.

The matrix A may be decomposed as follows.

$$A = D - L - U \tag{3.5}$$

where D consists of the diagonal blocks of A

L consists of the sub-diagonal blocks of A

U consists of the super-diagonal blocks of A.

The relationship between the splitting (3.3) and the decomposition (3.5) of the matrix A governs which of the classical block iterative methods is used.

3.2.1 Block Jacobi Method

If the relationship between the splitting and the decomposition of the matrix is

$$M = D \quad , \quad N = L + U$$

then (3.4) becomes the block Jacobi (BJ) method.

$$D\underline{x}^{k+1} = (L+U)\underline{x}^k + \underline{b}$$

Using the notation from (3.1), applying this iteration technique to the i^{th} row of blocks gives

$$A_i\underline{x}_i^{k+1} = \underline{b}_i - (B_{i-1}^T\underline{x}_{i-1}^k + B_i\underline{x}_{i+1}^k) \qquad i = 1,..,s \tag{3.6}$$

The finite element approximation for the examples used in this dissertation gives rise to strictly diagonally dominant diagonal blocks, A_i, hence these are invertible and each of the i block matrix equations (3.6) has a unique solution.

This method is the simplest, and slowest to converge, of all the block iterative methods. It has recently returned to favour with the widespread commercial availability of parallel computers, since each of the i sets of equations can be solved independently of the others.

3.2.2 Block Gauss-Seidel Method

If

$$M = D - L \quad , \quad N = U$$

then (3.4) becomes the block Gauss-Seidel (BGS) method.

$$(D - L)\underline{x}^{k+1} = U\underline{x}^{k} + \underline{b}$$

Applying this iteration technique to the i^{th} row of blocks in (3.1) gives

$$A_i\underline{x}_i^{k+1} = \underline{b}_i - (B_i\underline{x}_{i+1}^{k} + B_{i-1}^{T}\underline{x}_{i-1}^{k+1}) \qquad i = 1, .., s \tag{3.7}$$

Since the block tridiagonal matrix is a consistently ordered p-cyclic matrix (Section 3.1) and the diagonal blocks are non-singular then, from [13], if λ is a non-zero eigenvalue of the BGS iteration matrix ($\mathcal{L}_1$) and $\mu^2 = \lambda$, then μ is an eigenvalue of the BJ iteration matrix, B. Thus the BJ method converges if and only if the BGS iteration method converges, and if both converge then

$$\rho(\mathcal{L}_1) = \{\rho(B)\}^2 < 1$$

and consequently

$$R_\infty(\mathcal{L}_1) = 2R_\infty(B) \tag{3.8}$$

so in order to achieve the same accuracy, roughly twice as many iterations of the BJ method are required as for the BGS method.

The BGS method (3.7) is much less amenable to parallelisation than the BJ method because the right-hand side has a term from the current iteration, hence the i block systems must be solved in sequence.

3.2.3 Block Successive Over-Relaxation Method

If the solutions from the previous iteration, $\underline{x}_k$, and the latest iteration, $\underline{\tilde{x}}^{k+1}$, in the BGS method are combined with the use of an over-relaxation parameter, ω, then the result is the next iterate in the block successive over-relaxation (BSOR) method, $\underline{x}^{k+1}$ i.e.

$$\underline{x}^{k+1} = (1-\omega)\underline{x}^{k} + \omega\underline{\tilde{x}}^{k+1}$$

This is equivalent to the relationships

$$M = \frac{1}{\omega}(D - \omega L) \quad , \quad N = \frac{1}{\omega}\{(1-\omega)D + \omega U\}$$

so applying this iteration technique to the i^{th} row of blocks in (3.1) gives

$$A_i\underline{x}_i^{k+1} = \omega\underline{b}_i - \{(1-\omega)A_i\underline{x}_i^{k} + \omega(B_i\underline{x}_{i+1}^{k} + B_{i-1}^{T}\underline{x}_{i-1}^{k+1})\} \qquad i = 1,..,s \qquad (3.9)$$

If the value of the over-relaxation parameter is set to one, i.e. $\omega = 1$, then the block successive over-relaxation method is the same as the BGS method. As with the BGS method, this method is not easily parallelised because the right-hand side has a term from the current iteration so the i block systems must be solved in sequence.

From [13], if A is a consistently ordered p-cyclic matrix then the corresponding BSOR iteration will only converge if

$$0 < \omega < \left(\frac{p}{p-1}\right)$$

so since a block tri-diagonal matrix is a consistently ordered 2-cyclic matrix then the BSOR method (3.9) will only converge if

$$0 < \omega < 2$$

The value of ω which gives the fastest rate of convergence is ω_{opt}, i.e.

$$\min_{\omega}\{\rho(\mathcal{L}_{\omega})\} = \rho(\mathcal{L}_{\omega_{opt}})$$

where $\mathcal{L}_{\omega}$ is the BSOR iteration matrix. Since BGS is a special case of BSOR then

$$\rho(\mathcal{L}_{\omega_{opt}}) \leq \rho(\mathcal{L}_1)$$

i.e.

$$R_\infty(\mathcal{L}_{\omega_{opt}}) \geq R_\infty(\mathcal{L}_1) \tag{3.10}$$

Combining (3.8) and (3.10),

$$R_\infty(\mathcal{L}_{\omega_{opt}}) \geq R_\infty(\mathcal{L}_1) > R_\infty(B) \tag{3.11}$$

i.e. BSOR (with the optimum over-relaxation parameter) converges at least as fast as BGS, and both converge faster than BJ.

3.3 Block Preconditioned Conjugate Gradient Method

The conjugate gradient (CG) method for solving (3.2) when A is symmetric and positive definite is introduced by considering the minimisation of the functional,

$$\phi(\underline{x}) = \frac{1}{2}\, \underline{x}^T A\, \underline{x} - \underline{x}^T \underline{b} \tag{3.12}$$

The minimum of this functional is $-\frac{1}{2}\, \underline{b}^T A^{-1}\, \underline{b}$ and is achieved by setting $\underline{x} = A^{-1}\underline{b}$.

Proof :

$$\begin{aligned} \phi(A^{-1}\underline{b}) &= \tfrac{1}{2}((A^{-1}\underline{b})^T A (A^{-1}\underline{b})) - (A^{-1}\underline{b})^T \underline{b} \\ &= \tfrac{1}{2}(\underline{b}^T A^{-1} A A^{-1} \underline{b}) - (\underline{b}^T A^{-1} \underline{b}) \\ &= -\tfrac{1}{2}\, \underline{b}^T A^{-1}\, \underline{b} \end{aligned}$$

$$\begin{aligned} \text{So } \phi(\underline{x}) - \phi(A^{-1}\underline{b}) &= \tfrac{1}{2}\, \underline{x}^T A\, \underline{x} - \underline{x}^T \underline{b} - \tfrac{1}{2}\, \underline{b}^T A^{-1}\, \underline{b} \\ &= \tfrac{1}{2}(\underline{x} - A^{-1}\underline{b})^T A (\underline{x} - A^{-1}\underline{b}) \end{aligned}$$

Since A is positive definite then

$$(\underline{x} - A^{-1}\underline{b})^T A (\underline{x} - A^{-1}\underline{b}) \geq 0$$

so $\phi(x)$ is minimised when

$$(\underline{x} - A^{-1}\underline{b})^T A(\underline{x} - A^{-1}\underline{b}) = 0$$

$$\text{i.e.} \qquad \underline{x} = A^{-1}\underline{b}$$

and the minimum value of the functional is

$$\phi(\underline{x}) = -\frac{1}{2}\, \underline{b}^T A^{-1}\, \underline{b} \quad \Box$$

Thus minimising the functional (3.12) and solving (3.2) are equivalent problems. It is possible to minimise $\phi(\underline{x})$ by the steepest descent method [5]. At a point, $\underline{x}^k$, the functional ϕ^k $\;(= \phi(\underline{x}^k)\ )$ decreases most rapidly in the direction of $-\nabla\phi^k$ which is the residual, $\underline{r}^k = \underline{b} - A\underline{x}^k$. The new iterate, $\underline{x}^{k+1}$, is given by

$$\underline{x}^{k+1} = \underline{x}^k + \alpha\underline{r}^k \qquad\qquad \alpha \in \mathcal{R}$$

where α is chosen such that ϕ^{k+1} is minimised. This is achieved by setting

$$\frac{\partial\phi^{k+1}}{\partial\alpha} = 0$$

which gives

$$\alpha = \frac{(\underline{r}^k)^T\underline{r}^k}{(\underline{r}^k)^T A\underline{r}^k}$$

However, if the condition number of A, $\kappa(A)$, is large then the level curves of ϕ are very elongated hyperellipsoids and minimisation corresponds to finding the lowest point on a relatively flat, steep-sided valley; in the steepest descent method, we are forced to traverse back and forth across the valley rather than down the valley, the gradient directions that arise during the iteration are close thus making the progress towards the minimum point slow.

To overcome this problem, we successively minimise ϕ along a set of search directions $\{\underline{p}^0, \underline{p}^1, \ldots\}$ which do not necessarily correspond to the residual vectors $\{\underline{r}^0, \underline{r}^1, \ldots\}$. Then

$$\underline{x}^{k+1} = \underline{x}^k + \alpha\underline{p}^k$$

and again ϕ^{k+1} is minimised when

$$\frac{\partial \phi^{k+1}}{\partial \alpha} = 0$$

which gives

$$\alpha = \frac{(\underline{p}^k)^T \underline{r}^k}{(\underline{p}^k)^T A \underline{p}^k}$$

In order to ensure a reduction in the size of ϕ, $\underline{p}^k$ must not be orthogonal to $\underline{r}^k$,

$$\text{i.e.} \quad (\underline{p}^k)^T \underline{r}^k \neq 0$$

and the search directions are taken as A-conjugate to all the previous search directions in order to ensure convergence [5].

$$\text{i.e.} \quad P_{k-1}^T A \underline{p}^k = 0 \quad \text{where} \quad P_{k-1} = \{\underline{p}^0, \ldots, \underline{p}^{k-1}\} \in \mathcal{R}^{n \times k}$$

This is the basis of the CG method. From [5], the required search directions, $\underline{p}^k$, are found by taking

$$\underline{p}^k = \underline{r}^k + \beta \underline{p}^{k-1}$$

where

$$\beta = \frac{(\underline{r}^k)^T \underline{r}^k}{(\underline{r}^{k-1})^T \underline{r}^{k-1}}$$

3.3.1 Convergence

In exact arithmetic, the solution is obtained in at most n steps so the method has the property of finite termination (and would then be considered a direct method as opposed to an iterative method). However, in the practical situation of finite precision arithmetic, rounding errors lead to a loss of orthogonality among the residuals and finite termination is not mathematically guaranteed.

In any case, when the CG method is applied, n is usually so big that $O(n)$ iterations represents an unacceptable amount of work. Hence it is customary to regard the method as a genuinely iterative technique with termination based upon an iteration maximum, k_{max} and the residual norm.

From [5] an error bound on the CG method can be obtained in terms of the A-norm which is defined as follows,

$$\| \underline{w} \|_A = \sqrt{\underline{w}^T A \underline{w}}$$

After k iterations, a bound on the error is given by

$$\| \underline{x} - \underline{x}^k \|_A \quad \leq \quad 2 \| \underline{x} - \underline{x}^0 \|_A \left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1} \right)^k \tag{3.13}$$

where $\kappa_2(A)$ is the condition number of the matrix A with the underlying 2-norm (Section 3.1). In practice, this bound is too pessimistic and the accuracy of the $\{\underline{x}^k\}$ is often much better than this predicts. However, it is useful to note from (3.13) that the CG method converges very fast in the A-norm if $\kappa_2(A) \approx 1$.

3.3.2 Pre-conditioning

The main idea in pre-conditioning is to convert (3.2) into a related

$$\tilde{A}\tilde{\underline{x}} = \tilde{\underline{b}} \tag{3.14}$$

problem with $\tilde{A}$ being close to the identity so that $\kappa_2(\tilde{A})$ is close to unity. The pre-conditioner, $M \in \mathcal{R}^{n \times n}$, is symmetric and positive definite. Pre-multiplying the original system by M^{-1} gives

$$M^{-1} A \underline{x} = M^{-1} \underline{b}$$

so

$$\tilde{A} = M^{-1} A \qquad \tilde{\underline{x}} = \underline{x} \qquad \tilde{\underline{b}} = M^{-1} \underline{b}$$

and the standard CG method is applied to (3.14). The requirements for M are that

- M^{-1} is a good approximation to A^{-1} so that $\tilde{A}$ is "close" to the identity
- M is computationally simple and inexpensive to invert due to the occurrence of M^{-1} in the preconditioned CG algorithm.

In this dissertation, the pre-conditioner is taken as the diagonal blocks of the matrix A to enable the BCG method to be compared to the classical block iterative methods (which can be viewed as having the same diagonal block pre-conditioner) in Chapter 5.

3.3.3 Pre-conditioned Block Conjugate Gradient Algorithm

Applying the preconditioned CG method to the block tri-diagonal structure (3.1) leads to the following pre-conditioned block conjugate (BCG) algorithm.

```
$\underline{p}_i^0 = (M^{-1}\underline{r}^0)_i = (M^{-1}(\underline{b} - A\underline{x}))_i \qquad i = 1,\ldots,s$
$k = 0$
while $\| \underline{r}^k \| \neq 0$ and $k < k_{max}$

    $k = k + 1$
    $\alpha_k = \dfrac{\sum_{i=1}^{s} (\underline{r}_i^k)^T (M^{-1}\underline{r}^k)_i}{\sum_{i=1}^{s} (\underline{p}_i^k)^T (A\underline{p}^k)_i}$
    $\underline{x}_i^{k+1} = \underline{x}_i^k + \alpha_k \underline{p}_i^k \qquad i = 1,\ldots,s$
    $\underline{r}_i^{k+1} = \underline{r}_i^k - \alpha_k (A\underline{p}^k)_i \qquad i = 1,\ldots,s$
    $\beta_k = \dfrac{\sum_{i=1}^{s} (\underline{r}_i^{k+1})^T (M^{-1}\underline{r}^{k+1})_i}{\sum_{i=1}^{s} (\underline{r}_i^k)^T (M^{-1}\underline{r}^k)_i}$
    $\underline{p}_i^{k+1} = (M^{-1}\underline{r}^{k+1})_i + \beta_k \underline{p}_i^k \qquad i = 1,\ldots,s$
```

Chapter 4

Investigation of Nodal Ordering

In this chapter, the effect of the nodal ordering on the matrix structure generated by the finite element approximation described in Chapter 2 is examined. Two different ordering approaches are investigated on a sample problem which is solved by a block Jacobi iteration (Chapter 3). Using the results of this investigation, some recommendations are given for the nodal ordering in the discretisation of the region.

4.1 Discretisation of Region

In order to investigate the effect of the orientation of the node numbering scheme with respect to the predominant direction of flow, two different ordering approaches are used.

The first is a vertical slicing method. In this method, the region is divided by equally spaced vertical slices; then the vertical block sub-regions between pairs of adjacent slices are divided into equally sized rectangular prism elements. The nodes in the region are numbered on each slice in turn by natural ordering with the incrementation being faster in the vertical direction. An example of this is shown in Figure 4.1.

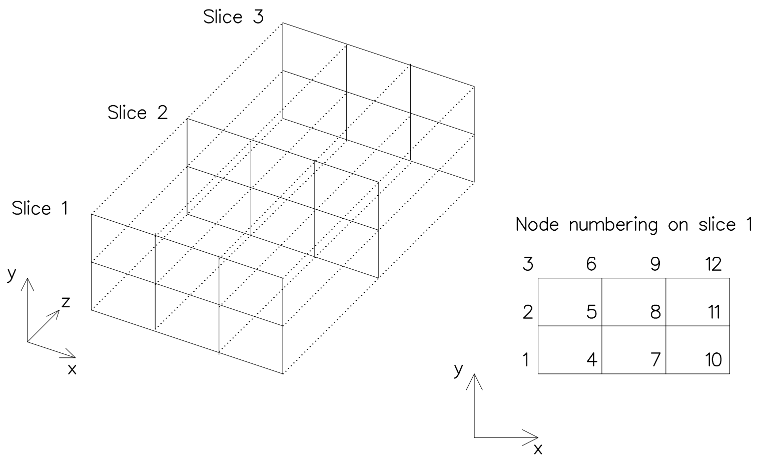

Figure 4.1: Vertical Slicing Nodal Ordering Using $3 \times 2 \times 2$ Elements

The other ordering approach is a horizontal slicing method. The procedure is essentially the same as for the vertical slicing method except that the slices are taken horizontally and the nodes on each slice are numbered in natural ordering with the incrementation being faster in the z-horizontal direction, as shown in Figure 4.2.

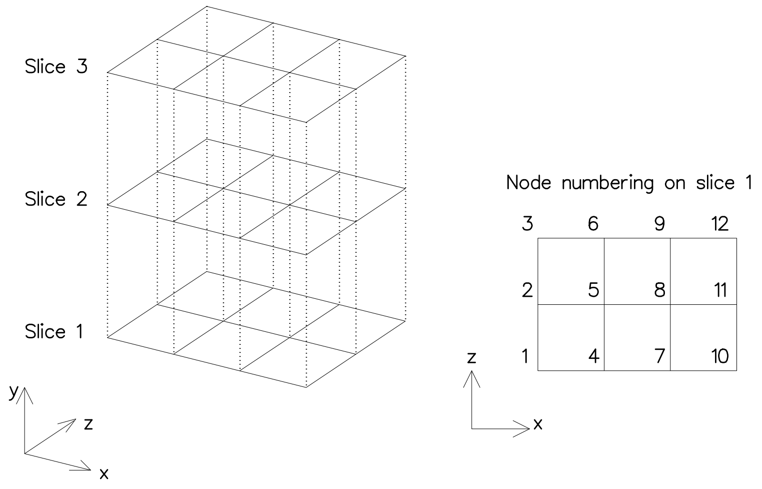

Figure 4.2: Horizontal Slicing Nodal Ordering Using $3 \times 2 \times 2$ Elements

4.2 Investigation of Convergence

The system of equations (2.3) is non-linear. In order that this system can be solved by matrix methods, a non-linear iteration technique is used. The system is linearised about the latest non-linear iterate to generate the matrix system to be solved.

The area of interest in these tests is the solution of the linearised system. Hence the non-linear iteration is not performed to convergence, only a single non-linear iteration being used and the convergence of the inner iteration method (used to solve the linearised system) is examined.

Due to the slicing methods used for the nodal ordering in the region, the global seepage matrix will have a symmetric and block tridiagonal structure as shown in (3.1). The connection between nodes on a slice in the approximate equation (2.3) appears as an entry in the diagonal blocks. The connection between nodes on adjacent slices appears as an entry in the super- and sub-diagonal blocks. If the direction of flow is in the direction of the slicing then the entries in the diagonal blocks will be large relative to those in the off-diagonal blocks and vice-versa.

The problem described in Section 2.3 is selected because it is an example of unsaturated flow. From [1], flow in the unsaturated region is predominantly in the vertical direction (as opposed to saturated flow which is predominantly in the horizontal direction). Hence, the block diagonal dominance should be stronger with the vertical slicing and so the convergence of block iterative methods should be faster with this slicing.

From (2.4), it can be seen that the two main influences on the relative sizes of the entries in the seepage matrix are K_{ij} and k_{rw}. The effects of these two are examined independently.

The region is discretised by the same number of elements in each direction i.e. $n \times n \times n$ elements. The resulting linearised matrix systems are solved by a block Jacobi iteration (Chapter 3). The initial pressure head iterate for the block Jacobi iteration is taken as -180 cm throughout the region. The criterion for convergence of this iteration method is that none of the entries in the most recent iterate change by more than 0.1 cm from the corresponding entry in the previous iterate.

4.2.1 Effect of K_{ij}

In order to eliminate the effects of k_{rw} from these tests, a constant initial (non-linear) iterate is taken to linearise the problem. This is

$$\psi_{initial} = -180 \text{ cm}$$

This ensures that k_{rw} is constant throughout the region during the block Jacobi iteration.

The hydraulic conductivity is taken to be anisotropic in the region. The components of K_{ij} are taken as

$$\begin{aligned} K_{xx} &= 1.0 \quad \text{cm/day} \\ K_{yy} &= 0.4 \quad \text{cm/day} \\ K_{zz} &= 0.1 \quad \text{cm/day} \\ K_{xy} &= 0.0 \quad \text{cm/day} \\ K_{yz} &= 0.0 \quad \text{cm/day} \\ K_{zx} &= 0.0 \quad \text{cm/day} \end{aligned}$$

The results of the tests are shown in Figure 4.3. This test case will be used to generate matrices in later parts of this dissertation; the test case with vertical slicing will be referred to as Problem 1 and the test case with horizontal slicing will be referred to as Problem 2.

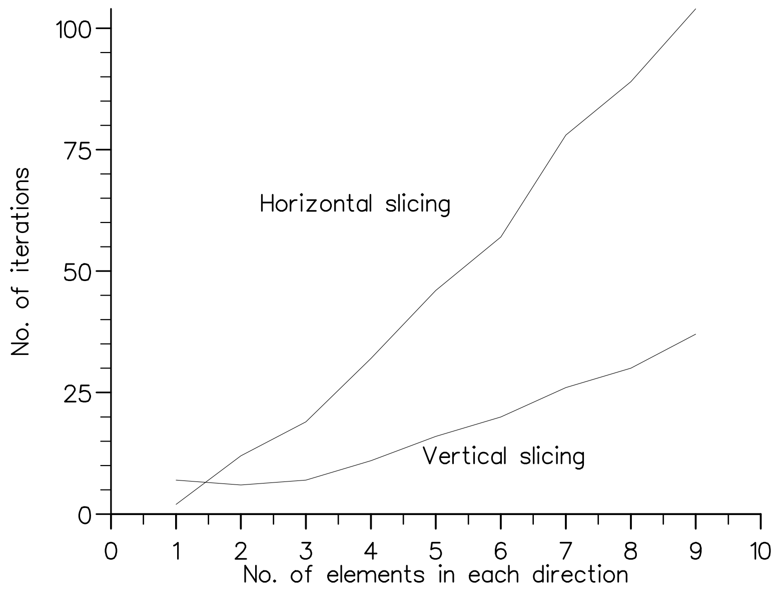

Figure 4.3: Effect of K_{ij} on number of iterations

The entries in the diagonal blocks for the vertical slicing depend on the x and y components of K_{ij} whereas they depend on the x and z components of K_{ij} for the horizontal slicing. Hence the vertical slicing should have a faster convergence rate, as borne out by Figure 4.3. It is a difficult task to measure the components of the saturated hydraulic conductivity tensor in practice so the porous medium is often taken to be largely isotropic. Also, unless there exists some form of discontinuous heterogeneity (e.g. faults or large scale stratigraphic features) or layered heterogeneity (caused by layers of individual beds each with its own homogeneous conductivity value), then regions are usually taken to be largely homogeneous. Under these isotropic and homogeneous conditions, there is no difference between the convergence rate given by the two slicing techniques on this problem.

4.2.2 Effect of k_{rw}

In order to eliminate the effect of K_{ij} in these tests, its components are taken as

$$\begin{aligned} K_{xx} &= 1.0 \quad \text{cm/day} \\ K_{yy} &= 1.0 \quad \text{cm/day} \\ K_{zz} &= 1.0 \quad \text{cm/day} \\ K_{xy} &= 0.0 \quad \text{cm/day} \\ K_{yz} &= 0.0 \quad \text{cm/day} \\ K_{zy} &= 0.0 \quad \text{cm/day} \end{aligned}$$

Since k_{rw} manifests itself as a non-linear effect, then a non-linear iteration must be performed to produce the second non-linear iterate (which more closely resembles the true form of the solution) and this is used to generate the linearised matrix system to be solved by the block iterative algorithm. The results of the tests are shown in Figure 4.4. This test case will be used to generate matrices in later parts of this dissertation; the test case with vertical slicing will be referred to as Problem 3 and the test case with horizontal slicing will be referred to as Problem 4.

From Figure 4.4 it can be seen that, again, the vertical slicing gives a faster rate of convergence. The difference in convergence rates is an indirect result of the boundary conditions imposed on the problem. For this problem, the boundary conditions cause the vertical flow to dominate. During the non-linear iteration, the flow from the previous iteration is used to calculate the linearised relative permeability throughout the region. The magnitude of this effect is governed by the non-linearity of the constitutive relations.

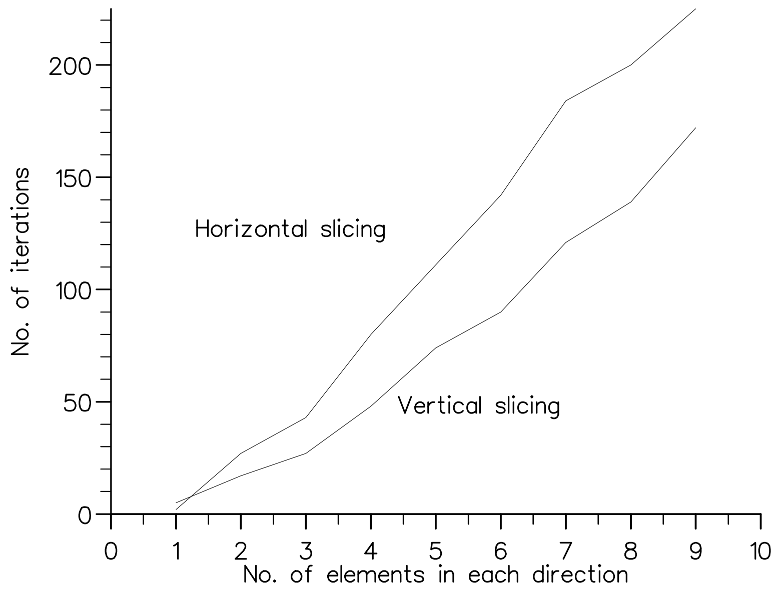

Figure 4.4: Effect of k_{rw} on number of iterations

4.3 Examples of Block Diagonal Dominance

Consider the $s \times s$ block tridiagonal system (3.1) with the inverse of the diagonal blocks used as a pre-conditioner. Using the notation of (3.5),

$$D^{-1}A\underline{x} = D^{-1}\underline{b}$$

$$\text{i.e. } (I - D^{-1}L - D^{-1}U)\underline{x} = D^{-1}\underline{b}$$

The degree of block diagonal dominance, σ, (see Section 3.1) of the pre-conditioned matrix, using the notation of (3.1), is

$$\sigma = \max_{i=1,.,s}\left\{\|I\|_\infty(\|A_i^{-1}B_{i-1}^T\|_\infty + \|A_i^{-1}B_i\|_\infty)\right\} \qquad B_0 = B_s = 0$$

and, since $\|I\|_\infty = 1$, then

$$\sigma = \max_{i=1,.,s}\left\{\|A_i^{-1}B_{i-1}^T\|_\infty + \|A_i^{-1}B_i\|_\infty\right\} \qquad B_0 = B_s = 0 \qquad (4.1)$$

Note that this is the norm of the block Jacobi iteration matrix, B, i.e.

$$B = D^{-1}(L+U) = D^{-1}L + D^{-1}U$$

$$\|B\|_\infty \le \|D^{-1}L\|_\infty + \|D^{-1}U\|_\infty$$

or, in the notation of (3.1),

$$\|B\|_\infty = \max_{i=1,\ldots,s} \left\{ \|A_i^{-1} B_{i-1}^T\|_\infty + \|A_i^{-1} B_i\|_\infty \right\} \qquad B_0 = B_s = 0$$

so, comparing with (4.1), it can be seen that

$$\|B\|_\infty = \sigma$$

Therefore the stronger the block diagonal dominance of the pre-conditioned matrix, the smaller the ∞-norm of the block Jacobi iteration matrix. It is conjectured that, for the types of matrices generated in this dissertation; the smaller the ∞-norm of the block iteration matrix, the smaller its spectral radius and hence the faster the asymptotic rate of convergence of the classical iteration methods.

The direct connection between the norm of the iteration matrix and its spectral radius is not proved in this dissertation, however some examples are given to support the belief that stronger block diagonal dominance should imply a greater asymptotic rate of convergence. The matrices resulting from Problems 1-4 (see Section 4.2) with $2 \times 2 \times 2$ elements used in the discretisation of the region (A_1, A_2, A_3, A_4 respectively) are given in Appendix B. The degree of block diagonal dominance, spectral radius of the corresponding block Jacobi matrix and number of iterations of the block Jacobi method to convergence, N, for these matrices are given in Table 4.1.

From this table it can be seen that the spectral radius of the Jacobi iteration matrix is smaller (and the number of iterations required for convergence is less) for stronger block diagonal dominance of the pre-conditioned matrix (i.e. smaller σ).

Matrices A_1 and A_3 are the result of vertical slicing node ordering approaches and matrices A_2 and A_4 are the result of horizontal slicing approaches. This confirms (at least numerically for these examples) that vertical slicing leads to

Matrix	σ	$\rho(B)$	N
A_1	0.8859	0.4382	6
A_2	0.9648	0.5957	12
A_3	0.9746	0.7215	17
A_4	1.0069	0.8147	27

Table 4.1: Degree of Block Diagonal Dominance for Example Matrices

stronger block diagonal dominance and faster convergence, as expected from the previous results in this chapter .

4.4 Recommendations for Nodal Ordering Approach

Plane slicing techniques (with the same number of nodes on each slice) are used in this project primarily because of coding convenience. As stated in [9], mesh grading capability is somewhat limited with this approach, particularly when the flow region contracts or expands in the direction normal to the plane of the slice.

In practice, a more general spatial discretisation will be required to model complex geometries and flows. The generalisations required would be to have

- a varying number of nodes on each slice
- various different types of geometry of element (e.g. tetrahedral)
- no requirement that the faces of an element which lie on adjacent slices be parallel.

With these generalisations (particularly the last one), the horizontal and vertical slicing terminology loses meaning. However the convergence concepts outlined in the previous sections still apply. In order that the following discussion has meaning, the term “slicing” will now be taken to mean a discretisation approach with a nodal ordering method which gives rise to a block tridiagonal matrix. The results of the previous section will be generalised to these more relaxed discretisation approaches.

From the previous sections, it can be seen that the direction of slicing can have a significant effect on the rate of convergence of block iterative methods applied to the linearised problem. This effect can be caused by heterogeneity and anisotropy of the material properties (e.g. the saturated hydraulic conductivity tensor) and also by the boundary conditions (manifesting themselves through the non-linear material characteristics).

The effect of the boundary conditions will be most pronounced in combined saturated-unsaturated flow where (as already stated in Section 4.3) flow is predominantly vertical above the water table in the unsaturated region and predominantly horizontal below the water table in the saturated region. Hence if the nodal ordering method employed is a single slicing technique over the whole region then this will lead to weaker convergence of a block iterative method in the part of the region where the flow is not aligned with the direction of slicing.

It is clear that different node ordering approaches should be used in different regions. The simplest and most obvious possibility is to use a vertical slicing approach above the water table and a horizontal one below it. However, consideration should also be given to the heterogeneity and anisotropy of material properties when choosing the ordering approach.

Chapter 5

Comparison of Block Matrix Iterative Methods

In this chapter, the performance of the block iterative methods described in Chapter 3 on the problems described in Chapter 4 is discussed.

5.1 Practical Application of Block Matrix Methods

It has already been stated that the matrices generated by the methods described in Chapters 2 and 4 give rise to symmetric block tridiagonal matrices (3.1). Hence only the diagonal and super-diagonal blocks need be generated and stored in the program.

Also, there is more structure to be exploited within each of the blocks because these are banded. From [5], the matrix $A = (a_{ij})$ has upper bandwidth q if $a_{ij} = 0$ whenever $j > i + q$ and lower bandwidth p if $a_{ij} = 0$ whenever $i > j + p$. For the matrices generated by the slicing techniques in Section 4.1,

$$q = p = N + 2$$

where N is the number of elements in the y-direction for vertical slicing, and the number of elements in the z-direction for horizontal slicing. This is not easily seen from the example matrices given in Appendix B because these are too small.

When applying the block iterative methods of Chapter 3, there is a matrix system corresponding to each of the s diagonal blocks to solve at each iteration. The only difference between these systems at each iteration is the right hand side vector so it is most efficient to compute the LDL^T decomposition of each of the diagonal blocks during the first iteration. Then each of the systems can be solved by forward substitution, division by the diagonal and backward substitution. Since the factors of a banded matrix are themselves banded then a banded LDL^T decomposition algorithm and banded forward and backward solvers can be used for these computations (e.g. those given in [5]).

5.2 Performance of Block Matrix Methods

In this section the block iterative methods are applied to the matrix systems arising from Problems 1-4 (Chapter 4) with $n \times n \times n$ elements used in the discretisation of the region. Both the BSOR method with $\omega = 1.5$ and with $\omega = \omega_{opt}$ (with ω_{opt} found to 2 decimal places by repeated running of the BSOR algorithm with different values of ω) are included.

The initial pressure head iterate is taken as -180.0 cm. The criterion for convergence is that none of the entries in the most recent iterate change by more than 0.1 cm from the corresponding entry in the previous iterate.

The number of iterations to convergence for each of the four problems is given in the four Tables later in this section. The data for Figures 4.3,4.4 and Table 4.1 in the previous chapter is given in the BJ columns in these tables.

If the BJ, BGS and BSOR(ω_{opt}) columns are compared in the four tables, it can be seen that the BGS method converges more quickly than the BJ method and that the BSOR(ω_{opt}) method converges at least as fast as the BGS method as predicted by (3.11).

The BSOR method with $\omega = 1.5$ converges faster than BGS for the larger cases in all four problems indicating that the matrix systems given by these problems are relatively insensitive to the choice of over-relaxation parameter and that BSOR is more effective than BGS for a reasonable choice of ω.

The methods which converge fastest are BSOR(ω_{opt}) and BCG. For Problems 1 and 2, BSOR(ω_{opt}) converges slightly faster than BCG while for Problems 3 and 4, both methods require approximately the same number of iterations for convergence.

If the BSOR and BCG algorithms (Chapter 3) are examined, it can be seen that during an iteration, apart from the shared computations (LDL^T decomposition, forward and backward solver, etc.), BCG requires an additional matrix-vector multiplication, two additional vector-vector multiplications and three additional vector-vector additions. Hence BCG requires more work per iteration than BSOR.

However, the use of BSOR(ω_{opt}) requires ω_{opt} to be determined. This can either be done by an a priori method (e.g. estimating the spectral radius of the corresponding Jacobi iteration matrix which is related to ω_{opt}) or adaptively (e.g. the adaptive SOR algorithm given in [6] which uses the difference between successive iterates to repeatedly refine ω towards ω_{opt} during the iteration). The BCG method does not require any parameter to be supplied.

If the number of extra calculations required by the BSOR method to calculate ω_{opt} is greater than the number of extra calculations required by the BCG method then the BCG method will be the most effective. However, if ω_{opt} can be calculated relatively cheaply then the BSOR method will be the most effective.

It is noted that if the results from Problem 1 and Problem 2, and the results from Problem 3 and Problem 4, are compared for each of the block iteration methods, then the matrices arising from the vertical slicing node ordering converge faster than those from the horizontal slicing nodal ordering, showing that the results of Chapter 4 apply to all the iteration methods and not just the block Jacobi method.

Size of Problem	Number of iterations				
$n \times n \times n$	BJ	BGS	BSOR $\omega = 1.5$	BSOR(ω_{opt}) $\omega = \omega_{opt}$	BCG
2	6	4	11	4 (1.00)	4
3	7	5	13	5 (1.00)	5
4	11	8	12	6 (1.11)	7
5	16	11	13	7 (1.19)	8
6	20	13	14	8 (1.23)	9
7	26	17	14	9 (1.30)	10
8	30	20	15	10 (1.33)	11
9	37	24	15	11 (1.39)	13

Table 5.1: Number of Iterations for Problem 1

Size of Problem	Number of iterations				
$n \times n \times n$	BJ	BGS	BSOR $\omega = 1.5$	BSOR(ω_{opt}) $\omega = \omega_{opt}$	BCG
2	12	8	12	5 (1.12)	6
3	19	12	13	7 (1.22)	8
4	32	20	12	9 (1.35)	11
5	46	29	13	12 (1.42)	14
6	57	35	15	13 (1.48)	16
7	78	46	18	16 (1.53)	19
8	89	54	21	17 (1.56)	21
9	104	66	27	18 (1.62)	24

Table 5.2: Number of Iterations for Problem 2

Size of Problem	Number of iterations				
$n \times n \times n$	BJ	BGS	BSOR $\omega = 1.5$	BSOR(ω_{opt}) $\omega = \omega_{opt}$	BCG
2	17	11	12	6 (1.20)	5
3	27	17	12	8 (1.29)	9
4	48	29	13	11 (1.43)	10
5	74	45	15	13 (1.53)	13
6	90	55	21	16 (1.57)	16
7	121	74	30	18 (1.64)	18
8	139	86	35	21 (1.66)	22
9	172	107	46	24 (1.69)	24

Table 5.3: Number of Iterations for Problem 3

Size of Problem	Number of iterations				
$n \times n \times n$	BJ	BGS	BSOR $\omega = 1.5$	BSOR(ω_{opt}) $\omega = \omega_{opt}$	BCG
2	27	14	12	7 (1.34)	6
3	43	24	12	9 (1.40)	9
4	80	43	15	13 (1.53)	12
5	111	64	25	16 (1.62)	15
6	142	76	31	18 (1.64)	18
7	184	99	42	21 (1.70)	21
8	200	112	48	23 (1.72)	24
9	225	136	61	27 (1.75)	27

Table 5.4: Number of Iterations for Problem 4

Chapter 6

Parallel Implementation of Block Matrix Algorithms

Block algorithms access data from memory in large chunks or blocks; in parallel computer architectures, input/output is usually more expensive than floating point computations, making block algorithms attractive for these machines. In this chapter, the possibility of parallel implementation of some of the block matrix methods described in Chapter 3 is discussed.

The block matrix algorithms are directed at a transputer type architecture. In the first part of this chapter, the transputer is described and some of the important properties required for the design of algorithms for this type of machine are identified.

6.1 Description of Transputer Architecture

6.1.1 The Transputer

A transputer (INMOS 1985) is a single chip microprocessor [7], designed as a building block for parallel processors. To facilitate this it has memory and four links to connect one transputer to another, all on a single VLSI chip. One important feature of VLSI technology is that communication between devices is very much slower than communication within a device. In transputer parts, all components execute concurrently; each of the four links and the floating point co-processor can all perform useful work while the processor is executing other instructions.

Transputers are designed to implement the parallel programming language OCCAM very efficiently. However, due to the large amount of resources residing in FORTRAN code, parallel FORTRAN is available, e.g. the 3L Parallel FORTRAN package [11].

6.1.2 Transputer Networks

The four links on a transputer allow several interconnection topologies to be implemented. Some examples of these are shown in Figure 6.1. The network topology of a system is a hardware feature. In a network, one transputer functions as the 'host', by which the network is directed. The host is installed in a PC and does not take part in the computational process itself.

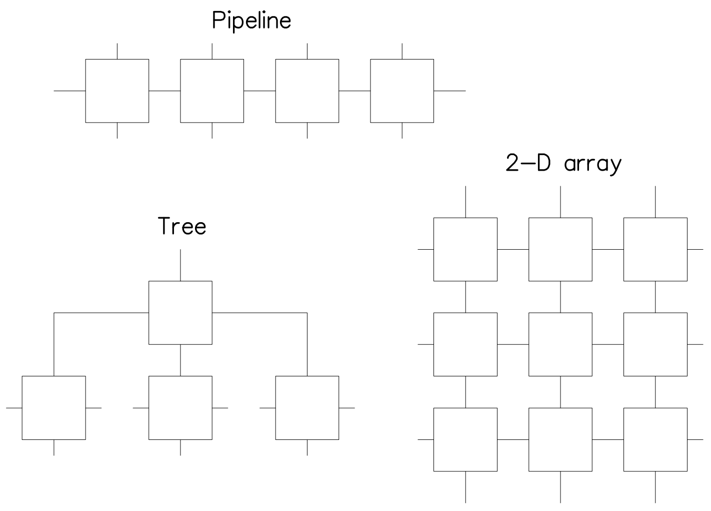

Figure 6.1: Transputer Network Topologies

The macroscopic structure of a parallel computer can be classified according to *Flynn's taxonomy* [7]. This system is based on the way in which the machine relates its instructions to the data being processed. If the 2-D array network topology is used, then the transputer system resembles a *Single Instruction-stream Multiple Data-stream* (SIMD) architecture in which several processors simultaneously execute the same instruction on multiple data streams (e.g. the ICL DAP). This type of architecture is represented in Figure 6.2, which appears amenable to block matrix algorithms.

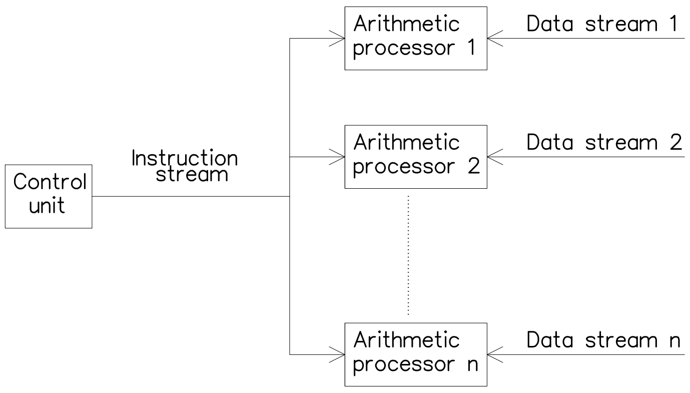

Figure 6.2: Single Instruction-stream Multiple Data-stream

However, it is possible to use a pipeline of transputers in a pseudo SIMD mode. This is the type of architecture considered for the implementation of the block matrix algorithms, since the hardware configuration of the available transputers is a pipeline. Note that an open-ended n-stage pipeline network has $(2n+1)$ free (or unused) links and therefore does not make optimum use of the transputer communication facilities. The processors communicate via access to an adjacent processors local memory (see Figure 6.3). This type of communication is inefficient with a pipeline network since data required by a transputer from a non-adjacent transputer must be passed along the chain through the intermediate transputers.

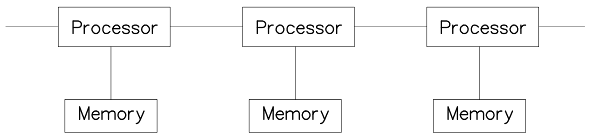

Figure 6.3: Communication Between Processors Via Local Memory

The pipeline will be implemented as a *systolic network* [5]. In a systolic network the processors operate in a completely synchronous fashion. During a "tick" of a global clock, each processor communicates with its neighbours and then performs some local computation.

6.2 Parallelisation of Block Matrix Algorithms

In this section, the possibility of implementation of the sequential block iterative methods described in Chapter 3 on a transputer system is discussed. It will be assumed that there are p transputers in the network (where p is less than the number of blocks on the diagonal, s).

6.2.1 Software Model

There are two main types of software parallelisation available in 3L Parallel FORTRAN [11], tasks and threads.

A task is a FORTRAN program. Task structure is static (i.e. tasks are not created and destroyed dynamically during execution) and no hierarchy exists (i.e. there is no facility for "sub-tasks" within tasks). There is no memory sharing between tasks, they communicate via channels which are one way communication paths with a fixed starting point and a fixed finishing point. A complete application is viewed as a collection of one or more tasks, a single transputer can execute more than one task. A task may contain several concurrent threads.

A thread is a FORTRAN subroutine. Threads are dynamically created and destroyed during execution and are potentially heirarchical (i.e. a thread may be created from within another thread). Threads have shared common blocks.

Since tasks are the main type of parallelisation in 3L Parallel FORTRAN, only these will be considered in the formulation of possible parallel algorithms. The task structure has the advantage that it is easy to adapt the application if more transputers are added to the system since the distribution of tasks on the available processors is controlled by a piece of software called the configuration file and it is relatively simple to alter this.

6.2.2 Block Jacobi Method

As stated in Section 3.2.1, this method is easily parallelised since each of the block matrix equations which result from the method are independent from the others. Hence each of the s block matrix equations can be coded as a separate task, and the s tasks can be distributed among the p processors. The algorithm for task i in the middle of the system, is given after the end of this paragraph. It will be assumed that initially, every task contains its own part of the initial iterate, $\underline{x}_{0,i}$, its own part of the right hand side vector, $\underline{b}_i$ and its own sub-, super- and diagonal blocks, B^T_{i-1}, B_i and A_i.

Block Jacobi Method : Task i Algorithm

```
1. $k = 0$

2. Send $\underline{x}_i^k$ to neighbouring tasks (i.e. task $i-1$ and task $i+1$).
```

3. Receive $\underline{x}_{i-1}^k$ from task $i-1$ and $\underline{x}_{i+1}^k$ from task $i+1$.

4. Modify the right hand side vector, i.e.

$$\tilde{\underline{b}}_i = \underline{b}_i - (B_{i-1}^T \underline{x}_{i-1}^k + B_i \underline{x}_{i+1}^k)$$

   using banded matrix-vector multiplication.

5. Perform a banded Cholesky decomposition on A_i i.e. $A = LDL^T$ (unless this has already been computed at a previous iteration).

6. Solve

$$LDL^T \underline{x}_i^{k+1} = \tilde{\underline{b}}_i$$

   for $\underline{x}_i^{k+1}$ by banded forward substitution, division by the diagonal and banded backward substitution.

7. $k = k + 1$

8. Return to step 2.

In the neighbouring tasks, the receive steps would have to be performed before the send step.

Note that no detail is given on the termination of the process. The termination criterion has to be global (i.e. the whole solution vector is tested for convergence at the same time) so all the results have to be assembled in one memory location at some stage during the iteration process and the convergence test is performed there.

One way of achieving this is to have all the transputers in the chain also connected to a single transputer (which itself is directly connected to the host processor). This enables the system to be tested for convergence at the end of each iteration, but means that the hardware configuration of the existing network has to be changed by adding extra wires and also limits the length of the chain to four transputers (including the host processor), hence this approach is rejected.

Another possibility is to pass all the results along the chain to a single transputer (where convergence is tested) after each iteration. However, this means that the bulk of the execution time of the program is consumed by the communication and so is not efficient.

Another approach is to pass the results along the chain to the first (non-host) processor where the convergence test is applied by a separate task - if convergence has occurred then this task informs the host to terminate the process and output the results. Since each transputer communicates with its neighbours at the end of each iteration then convergence is only tested after every $(p-1)$ iterations so some unnecessary iterations can be performed. In order to implement this approach, the block iteration task algorithm would have to be altered to incorporate the passing of data along the chain after each iteration; this is omitted from the task algorithm given for the block Jacobi algorithm so that the parallelism of the block iterative method is more clear (for this reason, no termination criterion will be given for the other parallel block iterative methods described in this chapter).

6.2.3 Block Gauss-Seidel Method

This method is not as easily parallelised since each of the block matrix equations uses the latest iterate from the previous block matrix equation. However, adapting the parallel point GS method from [10] for a block algorithm gives a possible approach for the parallel implementation of the BGS method.

Consider a general task in the middle of the chain of tasks. The previous task calculates its $(k+1)^{th}$ iterate and passes it to this task. This task then calculates its own $(k+1)^{th}$ iterate and passes it onto the next task. Up to this point there has been no departure from the sequential BGS method described in Section 3.2.2. However, the current task also passes its $(k+1)^{th}$ iterate back to the previous task which can now calculate its $(k+2)^{th}$ iterate. In this way, more than one task is active at any particular instant with different iteration levels being performed concurrently.

In this approach, some of the blocks will do extra iterations after the iteration which gives the solution to the required tolerance, so it appears that some processor time is wasted. However, in a parallel applications, the execution time is the chief measure of performance and, since the extra iterations are done at the same time as the required iterations then these extra iterations do not represent any waste in the parallel performance of the method. The algorithm for task i in the middle of the system, is given after the end of this paragraph; the same assumptions as for the BJ method in Section 6.2.2 are made here.

Block Gauss-Seidel Method : Task i Algorithm

1. $k = 0$
2. Receive $\underline{x}_{i-1}^{k+1}$ from task $i-1$.
3. Receive $\underline{x}_{i+1}^{k}$ from task $i+1$.
4. Modify the right hand side vector, i.e.

   $$\tilde{\underline{b}}_i = \underline{b}_i - (B_{i-1}^T \underline{x}_{i-1}^{k+1} + B_i \underline{x}_{i+1}^{k})$$

   using banded matrix-vector multiplication.
5. Perform a banded Cholesky decomposition on A_i i.e. $A = LDL^T$ (unless this has already been computed at a previous iteration).
6. Solve

   $$LDL^T \underline{x}_i^{k+1} = \tilde{\underline{b}}_i$$

   for $\underline{x}_i^{k+1}$ by banded forward substitution, division by the diagonal and banded backward substitution.
7. Send $\underline{x}_i^{k+1}$ to task $i+1$.

8. Send $\underline{x}_i^{k+1}$ to task $i-1$.

9. $k = k+1$

10. Return to step 2.

See Section 6.2.2 for the termination criterion for the method.

6.2.4 Block Successive Over-relaxation Method

The approach used for the parallelisation of the BGS method in the previous sub-section can also be used for the parallelisation of the BSOR method. The only difference in the task algorithm will be in step 4 which now reads as follows.

- Modify the right hand side vector, i.e.

$$\tilde{\underline{b}}_i = \omega \underline{b}_i - \{(1-\omega) A_i \underline{x}_i^k + \omega(B_i \underline{x}_{i+1}^k + B_{i-1}^T \underline{x}_{i-1}^{k+1})\}$$

  using banded matrix-vector multiplication.

6.2.5 Block Conjugate Gradient Method

If the sequential preconditioned BCG algorithm given in Section 3.3.3 is examined it can be seen that, during each iteration, the calculation of the parameters α and β require the results from all the blocks to be assembled in a single sum. To implement this algorithm in its current form on the type of transputer network available would cause the same difficulties as those described for the convergence test (see Section 6.2.2). Since the choices of α and β in the sequential algorithm ensure local minimisation of the functional over the space of A-conjugate search directions then any departure from these values will cause the method to lose this property. However, the iterated sequence generated by the BCG method with different choices of α and β may still converge to the solution of the system.

One possible approach is to have separate α and β for each block. In this case the algorithm becomes

```
$\underline{p}_i^0 = (M^{-1}\underline{r}^0)_i = (M^{-1}(\underline{b} - A\underline{x}))_i$        $i = 1, \ldots, s$
$k = 0$
while $\| \underline{r}^k \| \neq 0$ and $k < k_{max}$

        $k = k + 1$
        $\alpha_{k,i} = \frac{(\underline{r}_i^k)^T (M^{-1}\underline{r}^k)_i}{(\underline{p}_i^k)^T (A\underline{p}^k)_i}$        $i = 1, \ldots, s$
        $\underline{x}_i^{k+1} = \underline{x}_i^k + \alpha_{k,i}\underline{p}_i^k$        $i = 1, \ldots, s$
        $\underline{r}_i^{k+1} = \underline{r}_i^k - \alpha_{k,i}(A\underline{p}^k)_i$        $i = 1, \ldots, s$
        $\beta_{k,i} = \frac{(\underline{r}_i^{k+1})^T (M^{-1}\underline{r}^{k+1})_i}{(\underline{r}_i^k)^T (M^{-1}\underline{r}^k)_i}$        $i = 1, \ldots, s$
        $\underline{p}_i^{k+1} = (M^{-1}\underline{r}^{k+1})_i + \beta_{k,i}\underline{p}_i^k$        $i = 1, \ldots, s$
```

Note this this is no longer the conjugate gradient method but instead is a related method based on that procedure. This algorithm was found to converge only for the small test cases (e.g. 3 elements in each direction) indicating that the loss of conjugacy in this new method is too great.

The pre-conditioner, M, is the diagonal blocks of the original matrix; better performance may be obtained for this algorithm if the pre-conditioner contains some connection between adjacent block in the matrix e.g. the incomplete block Cholesky pre-conditioner given in [5] - this requires further investigation.

Another possible approach is one in which each task keeps a record of the most recent values of $\alpha_{k,i}$ and $\beta_{k,i}$ available to it for all the tasks and communicates its $\alpha_{k,i}$ and $\beta_{k,i}$ to its neighbours at each iteration. In this algorithm, values of $\alpha_{k,i}$ and $\beta_{k,i}$ from different iteration levels, k, would be in use at the same time - again, this requires further investigation.

Chapter 7

Conclusions

The matrix systems resulting from a finite element approximation (with slicing-type discretisation) to a three dimensional groundwater flow problem have been examined. It has been shown that the nodal ordering which leads to the fastest convergence rate of block iterative methods applied to these matrices is one in which the predominant direction of flow coincides with the direction of slicing.

When tested on example problems, the block successive over-relaxation method with optimum over-relaxation parameter, BSOR(ω_{opt}), and the block conjugate gradient method with diagonal block pre-conditioner, BCG, gave the fastest rate of convergence. Unless a cheap technique for determining the optimum over-relaxation parameter is used, BCG will be more efficient (even though BCG requires more floating point operations than BSOR).

It has been shown to be relatively simple to parallelise the classical block matrix iterative methods on a transputer system as long as modified convergence criteria are applied. However, since BCG involves some global iteration dependent parameters, then it cannot be parallelised in its current form on the type of transputer network available. Instead some modified "BCG" algorithms have been proposed but further investigation is required on these.

Appendix A

Influence Coefficients for Linear Rectangular Prism Element

A.1 Influence Coefficients for Element Seepage Matrix

$$[A^{xx}]^e = \frac{2}{3}\begin{pmatrix} a^{xx} & \frac{1}{2}a^{xx} \\ \frac{1}{2}a^{xx} & a^{xx} \end{pmatrix} \qquad [A^{yy}]^e = \frac{2}{3}\begin{pmatrix} a^{yy} & \frac{1}{2}a^{yy} \\ \frac{1}{2}a^{yy} & a^{yy} \end{pmatrix}$$

$$[A^{zz}]^e = \frac{1}{2}\begin{pmatrix} a^{zz} & -a^{zz} \\ -a^{zz} & a^{zz} \end{pmatrix} \qquad [A^{xy}]^e = \frac{2}{3}\begin{pmatrix} a^{xy} & \frac{1}{2}a^{xy} \\ \frac{1}{2}(a^{xy})^T & a^{xy} \end{pmatrix}$$

$$[A^{yz}]^e = \frac{1}{2}\begin{pmatrix} a^{yz} & \hat{a}^{yz} \\ (\hat{a}^{yz})^T & -a^{yz} \end{pmatrix} \qquad [A^{zx}]^e = \frac{1}{2}\begin{pmatrix} a^{zx} & \hat{a}^{zx} \\ (\hat{a}^{zx})^T & -a^{zx} \end{pmatrix}$$

where

$$a^{xx} = \frac{1}{6}\begin{pmatrix} 2 & -2 & -1 & 1 \\ -2 & 2 & 1 & -1 \\ -1 & 1 & 2 & -2 \\ 1 & -1 & -2 & 2 \end{pmatrix} \qquad a^{yy} = \frac{1}{6}\begin{pmatrix} 2 & 1 & -1 & -2 \\ 1 & 2 & -2 & -1 \\ -1 & -2 & 2 & 1 \\ -2 & -1 & 1 & 2 \end{pmatrix}$$

$$a^{zz} = \frac{1}{9}\begin{pmatrix} 4 & 2 & 1 & 2 \\ 2 & 4 & 2 & 1 \\ 1 & 2 & 4 & 2 \\ 2 & 1 & 2 & 4 \end{pmatrix} \qquad a^{xy} = \frac{1}{2}\begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & -1 & 0 & 1 \\ -1 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

$$a^{yz} = \frac{1}{3}\begin{pmatrix} 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & -2 & -1 \\ 0 & 0 & -1 & -2 \end{pmatrix} \qquad \hat{a}^{yz} = \frac{1}{3}\begin{pmatrix} 0 & 0 & -1 & -2 \\ 0 & 0 & -2 & -1 \\ 1 & 2 & 0 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix}$$

$$a^{zx} = \frac{1}{3}\begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & -2 & -1 & 0 \\ 0 & -1 & -2 & 0 \\ 1 & 0 & 0 & 2 \end{pmatrix} \qquad \hat{a}^{zx} = \frac{1}{3}\begin{pmatrix} 0 & -2 & -1 & 0 \\ 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 \\ 0 & -1 & -2 & 0 \end{pmatrix}$$

A.2 Influence Coefficients for Element Right Hand Side Vector

$$\{F^x\}^e = \left\{ \begin{array}{c} f^x \\ \\ f^x \end{array} \right\} \qquad \{F^y\}^e = \left\{ \begin{array}{c} f^y \\ \\ f^y \end{array} \right\} \qquad \{F^z\}^e = \left\{ \begin{array}{c} f^z \\ \\ -f^z \end{array} \right\}$$

where

$$f^x = \frac{mH}{2} \left\{ \begin{array}{c} -1 \\ 1 \\ 1 \\ -1 \end{array} \right\} \qquad f^y = \frac{lH}{2} \left\{ \begin{array}{c} -1 \\ -1 \\ 1 \\ 1 \end{array} \right\} \qquad f^z = \frac{lm}{2} \left\{ \begin{array}{c} -1 \\ -1 \\ -1 \\ -1 \end{array} \right\}$$

Appendix B

Small Example Matrices

In this appendix, the global seepage matrices arising from the test problems described in Chapter 4, with $2 \times 2 \times 2$ elements used to discretise the region, are given. (The entries in these matrices are only given to 2 decimal places.)

Matrix A_1 is the result of Problem 1 (Section 4.2.1)

Matrix A_2 is the result of Problem 2 (Section 4.2.1)

Matrix A_3 is the result of Problem 3 (Section 4.2.2)

Matrix A_4 is the result of Problem 4 (Section 4.2.2)

These matrices are used in Chapter 4 to illustrate the effect on the block diagonal dominance and asymptotic rate of convergence of different node ordering approaches. The matrices are also used to illustrate the structure of the global seepage matrices in Chapter 5.

$$A_1 = \left(\begin{array}{rrrrr|rrrrr|rrrrr}
0.08 & -0.01 & -0.02 & -0.03 & 0.00 & 0.03 & -0.01 & -0.01 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.01 & 0.16 & -0.03 & -0.03 & -0.03 & -0.01 & 0.05 & -0.02 & -0.03 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.02 & -0.03 & 0.16 & -0.02 & 0.00 & -0.01 & -0.02 & 0.05 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.03 & -0.03 & -0.02 & 0.31 & -0.02 & -0.02 & -0.03 & -0.02 & 0.11 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.03 & 0.00 & -0.02 & 0.16 & 0.00 & -0.02 & 0.00 & -0.02 & 0.05 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
\hline
0.03 & -0.01 & -0.01 & -0.02 & 0.00 & 0.16 & -0.02 & -0.03 & -0.07 & 0.00 & 0.03 & -0.01 & -0.01 & -0.02 & 0.00 \\
-0.01 & 0.05 & -0.02 & -0.03 & -0.02 & -0.02 & 0.31 & -0.07 & -0.06 & -0.07 & -0.01 & 0.05 & -0.02 & -0.03 & -0.02 \\
-0.01 & -0.02 & 0.05 & -0.02 & 0.00 & -0.03 & -0.07 & 0.31 & -0.04 & 0.00 & -0.01 & -0.02 & 0.05 & -0.02 & 0.00 \\
-0.02 & -0.03 & -0.02 & 0.11 & -0.02 & -0.07 & -0.06 & -0.04 & 0.63 & -0.04 & -0.02 & -0.03 & -0.02 & 0.11 & -0.02 \\
0.00 & -0.02 & 0.00 & -0.02 & 0.05 & 0.00 & -0.07 & 0.00 & -0.04 & 0.31 & 0.00 & -0.02 & 0.00 & -0.02 & 0.05 \\
\hline
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.03 & -0.01 & -0.01 & -0.02 & 0.00 & 0.08 & -0.01 & -0.02 & -0.03 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.01 & 0.05 & -0.02 & -0.03 & -0.02 & -0.01 & 0.16 & -0.03 & -0.03 & -0.03 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.01 & -0.02 & 0.05 & -0.02 & 0.00 & -0.02 & -0.03 & 0.16 & -0.02 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.02 & -0.03 & -0.02 & 0.11 & -0.02 & -0.03 & -0.03 & -0.02 & 0.31 & -0.02 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.02 & 0.00 & -0.02 & 0.05 & 0.00 & -0.03 & 0.00 & -0.02 & 0.16
\end{array}\right)$$

$$A_2 = \left(\begin{array}{rrrrrr|rrrrrr|rrr}
0.08 & 0.03 & 0.00 & -0.02 & -0.01 & 0.00 & -0.01 & -0.01 & 0.00 & -0.03 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.03 & 0.16 & 0.03 & -0.01 & -0.03 & -0.01 & -0.01 & -0.02 & -0.01 & -0.02 & -0.07 & -0.02 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.03 & 0.08 & 0.00 & -0.01 & -0.02 & 0.00 & -0.01 & -0.01 & 0.00 & -0.02 & -0.03 & 0.00 & 0.00 & 0.00 \\
-0.02 & -0.01 & 0.00 & 0.16 & 0.05 & 0.00 & -0.03 & -0.02 & 0.00 & -0.02 & -0.02 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.01 & -0.03 & -0.01 & 0.05 & 0.31 & 0.05 & -0.02 & -0.07 & -0.02 & -0.02 & -0.04 & -0.02 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.01 & -0.02 & 0.00 & 0.05 & 0.16 & 0.00 & -0.02 & -0.03 & 0.00 & -0.02 & -0.02 & 0.00 & 0.00 & 0.00 \\
\hline
-0.01 & -0.01 & 0.00 & -0.03 & -0.02 & 0.00 & 0.16 & 0.05 & 0.00 & -0.03 & -0.03 & 0.00 & -0.03 & -0.02 & 0.00 \\
-0.01 & -0.02 & -0.01 & -0.02 & -0.07 & -0.02 & 0.05 & 0.31 & 0.05 & -0.03 & -0.06 & -0.03 & -0.02 & -0.07 & -0.02 \\
0.00 & -0.01 & -0.01 & 0.00 & -0.02 & -0.03 & 0.00 & 0.05 & 0.16 & 0.00 & -0.03 & -0.03 & 0.00 & -0.02 & -0.03 \\
-0.03 & -0.02 & 0.00 & -0.02 & -0.02 & 0.00 & -0.03 & -0.03 & 0.00 & 0.31 & 0.11 & 0.00 & -0.02 & -0.02 & 0.00 \\
-0.02 & -0.07 & -0.02 & -0.02 & -0.04 & -0.02 & -0.03 & -0.06 & -0.03 & 0.11 & 0.63 & 0.11 & -0.02 & -0.04 & -0.02 \\
0.00 & -0.02 & -0.03 & 0.00 & -0.02 & -0.02 & 0.00 & -0.03 & -0.03 & 0.00 & 0.11 & 0.31 & 0.00 & -0.02 & -0.02 \\
\hline
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.03 & -0.02 & 0.00 & -0.02 & -0.02 & 0.00 & 0.16 & 0.05 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.02 & -0.07 & -0.02 & -0.02 & -0.04 & -0.02 & 0.05 & 0.31 & 0.05 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.02 & -0.03 & 0.00 & -0.02 & -0.02 & 0.00 & 0.05 & 0.16
\end{array}\right)$$

$$A_3 = \left(\begin{array}{rrrrr|rrrrr|rrrrr}
0.80 & -0.09 & 0.18 & -0.16 & 0.00 & -0.09 & -0.29 & -0.16 & -0.20 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.09 & 1.82 & -0.16 & 0.41 & -0.20 & -0.29 & -0.21 & -0.20 & -0.35 & -0.25 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.18 & -0.16 & 2.15 & -0.24 & 0.00 & -0.16 & -0.20 & -0.24 & -0.78 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.16 & 0.41 & -0.24 & 4.45 & -0.26 & -0.20 & -0.35 & -0.78 & -0.51 & -0.84 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.20 & 0.00 & -0.26 & 2.30 & 0.00 & -0.25 & 0.00 & -0.84 & -0.26 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
\hline
-0.09 & -0.29 & -0.16 & -0.20 & 0.00 & 1.60 & -0.18 & 0.36 & -0.31 & 0.00 & -0.09 & -0.29 & -0.15 & -0.20 & 0.00 \\
-0.29 & -0.21 & -0.20 & -0.35 & -0.25 & -0.18 & 3.63 & -0.31 & 0.83 & -0.39 & -0.29 & -0.21 & -0.20 & -0.35 & -0.25 \\
-0.16 & -0.20 & -0.24 & -0.78 & 0.00 & 0.36 & -0.31 & 4.28 & -0.49 & 0.00 & -0.15 & -0.20 & -0.24 & -0.78 & 0.00 \\
-0.20 & -0.35 & -0.78 & -0.51 & -0.84 & -0.31 & 0.83 & -0.49 & 8.89 & -0.52 & -0.20 & -0.35 & -0.78 & -0.50 & -0.84 \\
0.00 & -0.25 & 0.00 & -0.84 & -0.26 & 0.00 & -0.39 & 0.00 & -0.52 & 4.61 & 0.00 & -0.25 & 0.00 & -0.84 & -0.26 \\
\hline
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.09 & -0.29 & -0.15 & -0.20 & 0.00 & 0.79 & -0.09 & 0.18 & -0.15 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.29 & -0.21 & -0.20 & -0.35 & -0.25 & -0.09 & 1.81 & -0.15 & 0.41 & -0.20 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.15 & -0.20 & -0.24 & -0.78 & 0.00 & 0.18 & -0.15 & 2.13 & -0.24 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.20 & -0.35 & -0.78 & -0.50 & -0.84 & -0.15 & 0.41 & -0.24 & 4.44 & -0.26 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.25 & 0.00 & -0.84 & -0.26 & 0.00 & -0.20 & 0.00 & -0.26 & 2.31
\end{array}\right)$$

$$A_4 = \left(\begin{array}{rrrrrr|rrrrrr|rrr}
1.10 & -0.13 & 0.00 & 0.25 & -0.21 & 0.00 & -0.13 & -0.40 & 0.00 & -0.21 & -0.28 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.13 & 2.15 & -0.12 & -0.21 & 0.49 & -0.20 & -0.40 & -0.24 & -0.38 & -0.28 & -0.42 & -0.26 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.12 & 1.05 & 0.00 & -0.20 & 0.24 & 0.00 & -0.38 & -0.12 & 0.00 & -0.26 & -0.20 & 0.00 & 0.00 & 0.00 \\
0.25 & -0.21 & 0.00 & 1.65 & -0.19 & 0.00 & -0.21 & -0.28 & 0.00 & -0.19 & -0.60 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.21 & 0.49 & -0.20 & -0.19 & 3.25 & -0.18 & -0.28 & -0.42 & -0.26 & -0.60 & -0.37 & -0.58 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.20 & 0.24 & 0.00 & -0.18 & 1.60 & 0.00 & -0.26 & -0.20 & 0.00 & -0.58 & -0.18 & 0.00 & 0.00 & 0.00 \\
\hline
-0.13 & -0.40 & 0.00 & -0.21 & -0.28 & 0.00 & 1.86 & -0.21 & 0.00 & 0.42 & -0.36 & 0.00 & -0.15 & -0.19 & 0.00 \\
-0.40 & -0.24 & -0.38 & -0.28 & -0.42 & -0.26 & -0.21 & 3.63 & -0.20 & -0.36 & 0.83 & -0.34 & -0.19 & -0.29 & -0.18 \\
0.00 & -0.38 & -0.12 & 0.00 & -0.26 & -0.20 & 0.00 & -0.20 & 1.78 & 0.00 & -0.34 & 0.40 & 0.00 & -0.18 & -0.14 \\
-0.21 & -0.28 & 0.00 & -0.19 & -0.60 & 0.00 & 0.42 & -0.36 & 0.00 & 2.88 & -0.33 & 0.00 & -0.14 & -0.45 & 0.00 \\
-0.28 & -0.42 & -0.26 & -0.60 & -0.37 & -0.58 & -0.36 & 0.83 & -0.34 & -0.33 & 5.71 & -0.32 & -0.45 & -0.28 & -0.45 \\
0.00 & -0.26 & -0.20 & 0.00 & -0.58 & -0.18 & 0.00 & -0.34 & 0.40 & 0.00 & -0.32 & 2.83 & 0.00 & -0.45 & -0.14 \\
\hline
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.15 & -0.19 & 0.00 & -0.14 & -0.45 & 0.00 & 1.23 & -0.14 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.19 & -0.29 & -0.18 & -0.45 & -0.28 & -0.45 & -0.14 & 2.46 & -0.14 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & -0.14 & 0.00 & -0.45 & -0.14 & 0.00 & -0.14 & 1.23
\end{array}\right)$$

Bibliography

[1] D.M. Cooper (Institute of Hydrology), Private Communication, (1991)

[2] A.J. Davies, **The Finite Element Method : A First Approach**, OUP, (1986)

[3] **FLAMINCO Documentation : A Three-Dimensional Finite-Element Code for Analysis Water Flow and Solute Transport in Saturated/Unsaturated Porous Media**, Geo. Trans., Inc., (Sept. 1987)

[4] R.A. Freeze and J.A. Cherry, **Groundwater**, Prentice-Hall, (1979)

[5] G.H. Golub and C.F. Van Loan, **Matrix Computations**, 2nd Ed., John Hopkins University Press, (1989)

[6] L.A. Hageman and D.M. Young, **Applied Iterative Methods**, Academic Press, (1981)

[7] R.W. Hockney and C.R. Jesshope, **Parallel Computers 2 : Architecture, Programming and Algorithms**, Adam Hilger, Bristol, (1988)

[8] P.S. Huyakorn and B.M. Thompson, **Techniques for Making Finite Elements Competitive in Modelling Flow in Variably Saturated Porous Media**, Water Resources Research. 20, pp. 1099–1115, (Aug. 1984)

[9] P.S. Huyakorn, E.P. Springer, V. Guvanasen and T.D. Wadsworth, **A Three-Dimensional Finite Element Model for Simulating Water**

Flow in Variably Saturated Porous Media Flow Systems, Water Resources Research. 22, pp. 1790–1808, (Dec. 1986)

[10] J.J. Modi, **Parallel Algorithms and Matrix Computation**, Clarendon Press, Oxford, (1988)

[11] **Parallel FORTRAN User Guide**, 3L Ltd., Software Version 2.1.3, (Dec. 1990)

[12] G. Strang and G.S. Fix, **An Analysis of the Finite Element Method**, Prentice-Hall Int., (1973)

[13] R.S. Varga, **Matrix Iterative Analysis**, Prentice-Hall Int., London, (1962)